Diplomarbeit

Analyse, Entwurf und prototypische Implementierung eines sicheren Multi-Applikations-Frameworks für Chipkarten

Frank Puhlmann

Technische Universität Berlin
Fakultät IV - Informatik und Elektrotechnik
Fachgebiet Softwaretechnik

1. Gutachter: Prof. Dr. Stefan Jähnichen
2. Gutachter: Dr. Florian Kammüller

Betreuer: Dipl.-Informatiker Michael Nitsche

15. Dezember 2003

- rv1.1 -

Eidesstattliche Erklärung

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, den

_____

Unterschrift

Kurzfassung

In dieser Arbeit werden fortgeschrittene Techniken zur sicheren Rekonfiguration von Chipkarten untersucht und implementiert. Es wird eine weitgehend plattformunabhängige Lösung, mit heute erhältlichen Chipkarten und deren unterstützten Sicherheitsmerkmalen, erarbeitet. Die prototypische Implementierung ermöglicht die Rekonfiguration einer Chipkarte über offene Netze.

Nach Betrachtung der Grundlagen dieser Arbeit, Chipkarten, Netzwerke und IT-Sicherheit, werden relevante vorhandene Technologien aufgezeigt. Im nächsten Schritt werden Anforderungen für ein Multi–Applikations–Framework für Chipkarten diskutiert und evaluiert. Aufbauend auf den Anforderungen werden dann in der Systemanalyse verschiedene Anwendungsfälle entwickelt. Im Systementwurf werden diese zu Schnittstellen erweitert und weitere Komponenten wie Server und Client beschrieben. In der nachfolgenden prototypischen Implementierung wird der Entwurf in eine funktionsfähige Software umgesetzt. Ein abschließendes Kapitel beschreibt die Installation, Konfiguration und Demonstration der prototypisch entwickelten Software.

Durch die große Spannweite der Arbeit werden viele Probleme, Ideen und Anregungen für eine sichere Rekonfiguration von Chipkarten aufgezeigt. Das Ergebnis der Arbeit kann im Rahmen des Projektes „Campuskarte“ der TU–Berlin eingesetzt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Aufgabenstellung

Das TU-Projekt „Campuskarte“ befasst sich mit der Einführung elektronischer Ausweise (Chipkarten) für alle Hochschulangehörigen der Technischen Universität Berlin. Zu diesem Zweck wurde eine komplette Infrastruktur entworfen und implementiert. Diese basiert u.a. auf neuartigen Techniken, welche die Persönlichkeitsrechte der Nutzer besonders schützen. Als Chipkarte wird eine multifunktionale Java-Karte verwendet, welche Authentisierungs-, Signatur- sowie Verschlüsselungsfunktionalität bereitstellt. Mit Hilfe dieser Chipkarte können verschiedene Dienste wie Rückmeldung, sichere E-Mail oder Reisekostenabrechnung für Mitarbeiter genutzt werden.

Die momentan verwendete Chipkarte ist nach der Personalisierung nicht rekonfigurierbar. Es können weder Updates der auf der Karte vorhandenen Programme durchgeführt noch zusätzliche Programme installiert werden. Beides wäre jedoch in Anbetracht der geplanten Lebensdauer einer Chipkarte von fünf Jahren eine sehr nützliche Funktionalität. Die dafür benötigten Sicherheitsmerkmale und Speicherkapazitäten sind bedingt durch die technische Entwicklung jedoch erst in jüngster Zeit verfügbar.

Das Ziel der Diplomarbeit soll eine Erweiterung des vorhandenen Campuskarten Frameworks um die Möglichkeit der dynamischen Rekonfiguration der Chipkarte über offenen Netze sein.

Dazu sollen bestehende Multi-Applikations Lösungen für Chipkarten kritisch erörtert und diskutiert werden, um daraus neue und bestehende Gedanken in die Realisierung der Aufgabe einfließen zu lassen.

Ein wesentliches Augenmerk sollte auf den Entwurf und die Spezifikation der Multi-Applikations Erweiterung gelegt werden. Hierzu sind nicht nur neue, sondern auch vorhandene und zu erweiternde Systemteile zu berücksichtigen.

Aufgrund der Kartenmengen und individuellen Konfigurierbarkeit ist eine mögliche Integration der Chipkarte in ein flexibles Kartenmanagement zu betrachten. Entsprechende Erweiterungsmöglichkeiten sollen im Design berücksichtigt werden.

Das Konzept für die Realisierung soll unter Aspekten Authentizität, Vertraulichkeit und Integrität dem aktuellen Stand der Technik entsprechen.

Kapitel 1

Einleitung

1.1 Motivation und Ziel

Diese Arbeit wurde im Rahmen des TU-Projektes „Campuskarte“ erstellt. Im Laufe des Projektes wurde eine komplette Infrastruktur für Chipkarten entwickelt. Ein weißer Fleck auf der Roadmap ist die Integration eines Dienstes zur Rekonfiguration der Chipkarte nach der Ausgabe. In einer von Innovationen geprägten Branche, wie der Informatik, ist die gewünschte Lebensdauer einer Chipkarte von fünf Jahren eine schwere Herausforderung. Ein Multi–Applikations–Framework könnte wesentlich zur verbesserten Wertschöpfung der Campuskarte beitragen. Neue Anwendungen, welche ein entsprechendes Gegenstück auf der Chipkarte benötigen, würden ebenso ermöglicht wie das Aktualisieren vorhandener Programme auf der Karte. Auf ein zeitaufwendiges und teures Umtauschverfahren kann verzichtet werden. Dem Anwender wird eine neue Möglichkeit der Selbstbestimmung in die Hand gegeben, indem er entscheiden kann, welche Anwendungen und damit Funktionalitäten auf seiner Chipkarte zugelassen werden.

Den Mittelpunkt dieser Arbeit bildet eine Art „Machbarkeitsstudie“, mit dem Ziel neue Ideen und Anregungen für ein Multi–Applikations–Framework für Chipkarten zu erhalten. Das Aufgabengebiet umfasst die Entwicklung der Rekonfigurationsmöglichkeit einer Chipkarte über offene Netze. Weiterhin sind sämtliche dafür benötigte Komponenten zu ermitteln und zu beschreiben. Da dies ein sehr umfassendes Thema ist, werden verschiedene Teile nur insofern betrachtet, wie dies zur Erfüllung der Aufgabenstellung nötig ist. Gerade durch die Evaluation einer Vielzahl von Komponenten über verschiedene Stufen lassen sich jedoch sehr viele Ideen und Anregungen finden, die in späteren Arbeiten weiterverwendet werden können.

Wie sich erst im Laufe der Arbeit herausstellte, konnten Teile des vorhandenen Campuskarten–Frameworks nicht für eine Rekonfiguration der Chipkarte über offene Netze verwendet werden. Dadurch wurde diese Arbeit gleichzeitig zu einem „Redesign“ zentraler Komponenten der Campuskarteninfrastruktur, welche bisher lediglich zur Vorpersonalisierung der Chipkarte verwendet wurden. Die Analyse, der Entwurf sowie die prototypische Implementierung orientieren sich sehr stark an dem, mit heute erhältlichen Mitteln, technisch Machbaren. Somit beinhaltet diese Arbeit gleichzeitig einen Prototypen zum Erkennen von möglichen Problemen und Einschränkungen einer Rekonfiguration der Chipkarte nach der Ausgabe.

1.2 Überblick

Der Titel der Arbeit fordert die Analyse, den Entwurf und die prototypische Implementierung eines sicheren Multi–Applikations–Frameworks für Chipkarten. Die Betonung liegt auf dem Wort „sicheren“, wobei Sicherheit in der prototypischen Implementierung dem aktuellen Stand der Technik entspricht.

IT–Sicherheit ein umfassender Prozess, welcher sämtliche Bereiche der Entwicklung, des Betriebs und der Wartung von Software umfassen sollte. Dementsprechend enthält diese Arbeit kein separates Sicherheitskapitel, vielmehr werden in jedem Abschnitt entsprechende Betrachtungen vorgenommen. Sicherheit zieht sich als „roter Faden“ durch verschiedene Kapitel.

Die Arbeit beginnt mit der Zusammenfassung relevanter Grundlagen in den Bereichen Chipkarten, Netzwerke und IT–Sicherheit. Da Informatiker zur Erreichung einer Aufgabe sehr oft viele vorhandene Technologien integrieren müssen, wird ein Überblick über vorhandene Teillösungen gegeben.

Im Anschluss folgt eine Analyse, Entwurfs- und Implementierungsphase. Die hier beschriebene Darstellung wurde selbstverständlich erst nach vielen Iterationsphasen erreicht.

Komplettiert wird die Arbeit durch die Beschreibung der Installation, Konfiguration und Demonstration der prototypisch entwickelten Software. Diese Software, sowie die dazugehörige API–Dokumentation befinden sich auf der zu dieser Arbeit gehörenden CD.

Ein Kapitel mit einer Zusammenfassung und einem Ausblick auf zukünftige Entwicklungen schließt die Arbeit ab.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen der Diplomarbeit betrachtet. Dies sind in erster Linie Chipkarten sowie Netzwerktechnologien. Da die Chipkarte über offene Netze konfiguriert werden soll, müssen auch die Grundlagen der IT-Sicherheit betrachtet werden.

2.1 Chipkarten

Allgemeine Informationen zu Chipkarten befinden sich in [54]. Weitere Informationen sind in [21] zu finden.

2.1.1 Geschichte

Die Urahnen der heutigen Chipkarten, einfache Plastikkarten mit einer Hochprägung für Namen und Nummer, kamen bereits in den fünfziger Jahren des 20. Jahrhunderts in den USA auf. Die Sicherheit, dieser nur für privilegierte Menschen gedachten Karten, basierte auf der optischen Kontrolle der Karte sowie der Unterschrift auf der Rückseite; sie hing also von der Qualität der Prüfung in den einzelnen Akzeptanzstellen ab (gegenstandsbasierte Sicherheit). Die Möglichkeiten des bargeldlosen Zahlungsverkehrs mit dieser Art Karte wurde durch Unternehmen wie Visa und Mastercard vorangetrieben. Mit der weiteren Ausbreitung, zuerst in den USA, später in Europa und der restlichen Welt, stiegen jedoch auch die Probleme. Organisierte Kriminalität, Betrug und Zahlungsunfähigkeit von Kunden sind nur einige davon.

Zur Lösung wurde auf der Rückseite der Plastikkarte ein Magnetstreifen angebracht, der elektronisch gelesen und beschrieben werden konnte. Dieses Verfahren ermöglichte nicht nur den Wegfall der Papierbelege, sondern der Benutzer konnte auch durch neuartige Mechanismen authentisiert werden. Durchgesetzt als elektronische Unterschrift hat sich schließlich die persönliche Identifikationsnummer (PIN), die nur dem Benutzer bekannt ist und zur Durchführung einer Transaktion eingegeben werden muss (gegenstands- und wissensbasierte Sicherheit).

Jedoch hatten auch diese Karten sehr viele Nachteile. So ist es zum Beispiel möglich den Karteninhalt auszulesen, so dass eine PIN nicht sicher auf der Karte gespeichert werden kann. Die meisten Karten erlauben auch das beliebige ändern von Informationen auf dem Magnetstreifen, obwohl hier Techniken wie digitale Signaturen Abhilfe schaffen können (siehe Abschnitt 2.3). Weiterhin sind Magnetstreifenkarten „dumm“, sie können keine Angriffe erkennen und sich dagegen wehren. Die

Überprüfung persönlicher Merkmale muss durch Hintergrundsysteme wie Terminals oder Rechenzentren erfolgen.

Die rasanten Fortschritte in der Mikroelektronik ermöglichten es, Anfang der siebziger Jahre, Mikroprozessoren in die vorhandenen Plastikkarten zu integrieren. Diese boten den privaten Schlüsseln der ebenfalls in dieser Zeit entwickelten Public-Key-Kryptographie (siehe Abschnitt 2.3) eine sichere Unterkunft. Der Mikrochip auf der Karte kann sicherheitsrelevante Operationen direkt auf der Karte ausführen, ohne dass geheime Informationen die Karte verlassen müssen.

2.1.2 Kontaktbehaftete und kontaktlose Chipkarten

Chipkarten gibt es für verschiedene Anwendungsgebiete mit kontaktbehafteter oder kontaktloser Schnittstelle. Auch eine Kombination von beidem ist vorgesehen, wobei dann entweder zwei verschiedene Chips auf der Karte vorhandenen sind oder ein Chip zwei Schnittstellen nach außen besitzt.

Kontaktbehaftete Chipkarten müssen zur Verwendung in einen Chipkartenleser eingeführt werden, dabei können durch mechanische Abnutzung die Kontakte verschleißen und so die Chipkarte unbrauchbar machen. Kontaktbehaftete Chipkarten können dafür sehr komplexe Operationen ausführen, da ihnen direkt eine Spannung zugeführt wird.

Kontaktlose Chipkarten besitzen keine physikalisch nach außen geführte Schnittstelle. Stattdessen sind im Inneren der Karte mehrere induktive Koppelspulen vorhanden. Über diese wird die Karte vom Kartenleser mit Energie und Daten versorgt. Weitere Spulen dienen dem Senden von Informationen. Die Verbindung zwischen Karte und Kartenleser ist je nach Typ sehr instabil. Auch können mehrere Karten in die Reichweite des Kartenlesers gebracht werden. Kontaktlose Chipkarten können nur sehr einfache Operationen ausführen. Es gibt drei verschiedene Typen von kontaktlosen Chipkarten. Die Close Coupling Cards müssen zur Kontaktaufnahme in einen Leseschlitz eingeführt oder auf eine bestimmte Stelle des Kartenlesers gelegt werden. Sie bieten im wesentlichen den Vorteil der nicht vorhandenen mechanischen Abnutzung. Proximity Cards sehen eine Reichweite von ca. 10 cm vor. Diese werden sehr oft zur Zutrittskontrolle verwendet, da die Karte z.B. nicht mehr aus der Brieftasche entnommen werden muss. Eine Antikollisionerkennung verhindert hierbei, dass sich verschiedene Karten untereinander stören. Die Remote Coupling Cards (auch Hands Free Cards genannt) bieten eine Reichweite von bis zu einem Meter. Karten dieser Generation müssen nicht mehr aus der Hand- oder Jacketttasche entnommen werden, es reicht ein Vorbeilaufen am Kartenleser zur Kommunikation.

Je größer die Entfernung zwischen Kartenleser und kontaktloser Chipkarte wird, umso weniger Energie kann übertragen werden, womit zugleich die Komplexität der möglichen Operationen auf der Chipkarte sinkt.

2.1.3 Speicher- und Prozessorkarten

Chipkarten können von ihrer Funktionalität her in Speicher- und Prozessorkarten unterteilt werden, wobei die letzteren selbstverständlich auch Speicher enthalten. Speicherkarten bieten nach verschiedenen Methoden Datenbereiche an. Auf diesen können, mit bestimmten Schlüsseln, Operationen wie Inkrement und Dekrement durchgeführt werden können. Speicherkarten dienen z.B. als Telefonkarte, deren Wert nur verringert werden kann. Prozessorkarten arbeiten nach dem

Von-Neumann-Prinzip [15], womit sie einen kompletten Miniaturrechner darstellen. Alle vorstellbaren Programme können (unter Berücksichtigung von Rechenleistung und Speicher) auf einer Chipkarte ausgeführt werden, wobei die Vorteile wie geschützter Speicherbereich, leichte Transportabilität sowie Universalität erhalten bleiben. Chipkarten können zusätzlich spezielle Hardware wie Krypto-Prozessoren[1] oder Biometrie-Sensoren[2] enthalten.

2.1.4 Betriebssysteme für Chipkarten

Es gibt verschiedene proprietäre sowie offene Betriebssysteme für Chipkarten. Proprietäre Chipkarten können nur mit speziellen Entwicklertools des jeweiligen Herstellers programmiert werden. Die einfache Übertragung der erstellten Programme auf andere Chipkarten ist nicht möglich. Offene Betriebssysteme, wie z.B. Java Card, Basic Card oder Multos ermöglichen eine Portabilität von einmal erstellten Programmen auf andere Chipkarten. Zusätzlich enthalten z.B. Java-Karten auch die wesentlichen Vorteile der entsprechenden Sprache, wie Speicherschutz und Zugriffskontrolle.

2.2 Netzwerke

Die folgenden Informationen wurden aus [58] entnommen. Weitergehende Informationen befinden sich in [42].

2.2.1 Geschichte

Die Geschichte der Telekommunikation beginnt mit der Frage, wie Informationen schneller als der Mensch reisen können. Schon zu Urzeiten wurden visuelle und akustische Signalisierungen eingesetzt, um Informationen zu übermitteln. Dies reichte von Rauchzeichen über Buschtrommeln bis hin zum optischen Telegraphen von Claude Chappe[3].

Im Jahre 1837 erfand Samuel F. B. Morse den elektronischen Telegraphen und entwickelte den Morse-Kode[4]. Dank der digitalen Kodierung der übertragenen Informationen, auch über schlechte Leitungen und große Entfernungen, breitete sich der elektronische Telegraph sehr schnell aus und revolutionierte die Kommunikationsmöglichkeiten der damaligen Zeit.

Die Ära der analogen Sprachkommunikation begann mit den Erfindungen von Antonio Meucci um 1860, Philip Reis im Jahre 1861 sowie von Alexander Graham Bell 1876. Analoge Telefonsysteme übertrugen menschliche Sprache über einfache elektrische Leitungen, wodurch das System einer starken Dämpfung unterlag. Eine Übertragung über größere Entfernungen war nur in sehr schlechter Qualität möglich, da die zwischengeschalteten Verstärker auch alle Störungen verstärkten. Zur Lösung dieser Probleme trug die Einführung von digitalen Fernleitungen bei. Hierbei wurden

[1] Ein Krypto-Prozessor ist für kryptographische Operationen, wie Ver- und Entschlüsselung, zuständig.

[2] Biometrie-Prozessoren und -Sensoren verarbeiten biometrische Merkmale, wie z.B. Fingerabdrücke.

[3] Claude Chappe installierte die erste optische Übertragungsstrecke im Jahre 1794 von Paris nach Lille. Allerdings wurden keine Licht, sondern lediglich Winkersignale übertragen, welche an der jeweils nächsten Station gesichtet und manuell nachgestellt werden mussten.

[4] Der Morse-Kode ist u.a. hier beschrieben: *http://www.wikipedia.org/wiki/Morse_code*

die Informationen in den Vermittlungsstellen digitalisiert, übertragen und wieder in analoge Signale konvertiert. Da sich der Zustand eines digitalen Signales sehr viel besser von Rauschen und Störungen unterscheiden lässt als der eines analogen Signales, können die zwischengeschalteten Verstärker das digitale Signal ohne Verlust weiterleiten.

In den sechziger Jahren des 20. Jahrhunderts kam die digitale Datenkommunikation auf. Computer wandelten mit Hilfe von Modems die zu übertragenden Daten in analoge Tonsignale um und schickten diese zu anderen Computern. Später wurde die Kommunikation durch direkte digitale Datenübermittelung über Kupferkabel, Glasfaser oder Funk abgelöst.

Im Jahr 1969 wurde ein von der amerikanischen Regierung finanziertes Netzwerk mit dem Namen ARPANET eröffnet, welches, auch bei einem Atombombenangriff auf Teile des Netzes, funktionsfähig bleiben sollte. Erreicht wurde dies durch eine hohe Dezentralisierung. Sehr wichtige, weitere Vorteile sind größtmögliche Heterogenität der angeschlossenen Rechner mittels universeller Konnektivität sowie die Übermittelung von Datenpaketen.

2.2.2 Internet

Aus dem ARPANET entwickelte sich dann das uns heute bekannte Internet[5]. Das, was wir als Internet bezeichnen, ist der Zusammenschluss vieler unterschiedlicher Netzwerke, die von öffentlichen oder kommerziellen Geldgebern betrieben werden. Ein Beispiel eines Netzwerkes für den Bereich Wissenschaft, Forschung und Bildung in Deutschland ist das Deutsche Forschungsnetz (DFN)[6].

Alle Teilnehmer eines Netzwerkes kommunizieren über das Internet-Protokoll (IP) miteinander [23].

Das Internet-Protokoll beschreibt, wie ein System von verteilten Computern untereinander Datenpakete austauschen kann. Dazu bekommt jeder teilnehmende Computer eines Netzwerkes eine (zumindest im jeweiligen Netzwerk) eindeutige Nummer im Format $x.x.x.x \quad (x \in 0..255)$, welche IP-Adresse genannt wird. In gewöhnlichen Netzen besitzt jede physikalische Netzwerkschnittstelle eines Computers eine MAC-Adresse[7]. Eine oder mehrere IP-Adressen können auf eine MAC-Adresse abgebildet werden. Wenn jetzt ein Datenpaket von einem Computer mit der IP-Adresse x an den Computer mit der IP-Adresse y geschickt werden soll, so muss zunächst die MAC-Adresse des Computers mit der IP-Adresse y ermittelt werden. Dies geschieht über das so genannte Address Resolution Protocol (ARP). Nachdem die physikalische MAC-Adresse bekannt ist, wird ein Weg (Route) über verschiedene Knoten zum Ziel gesucht. Das Datenpaket wird dem Computer zugestellt und höhere Treiberschichten kümmern sich um die weitere Zuordnung entsprechend der IP-Adresse.

Da IP-Adressen für Menschen sehr schwer einprägbar sind, wurde ein System zum Übersetzen von IP-Adressen in Domain-Namen eingeführt [24]. Ein Domain-Name wird von rechts nach links gelesen und setzt sich folgendermaßen zusammen: `subdomain.domain.topleveldomain`. Die `topleveldomain` gibt ursprünglich die Länder- oder Gruppenkennung des Rechners an, z.B. `de` für Deutschland oder

[5] Die im Folgenden beschriebene, grundlegende Technologie des Internets wird auch in Teilnetzen, sogenannten Intranets verwendet.

[6] *http://www.dfn.de*

[7] MAC steht für Media Access Control, eine weltweit eindeutige Ziffernfolge.

`com` für kommerzielle Anbieter. Allerdings ist die DNS-Vergabe kein Indiz mehr für einen eindeutigen Standort oder eine Gruppe, da zumindest in Deutschland die meisten Endungen in Kombination mit einer Domain frei erworben werden können. Die `domain` gibt den eigentlichen Rechnernamen oder den Namen eines Subnetzes an. Die `subdomain` kann weitere `subdomains` enthalten und gibt dann den subnetz-internen Rechnernamen oder einen Dienstnamen (siehe Abschnitt Dienste) an. Verschiedene Computer im Internet, so genannte DNS–Server[8], bieten Übersetzungsdienste von Domain-Namen in IP-Adressen an.

Die Anzahl der möglichen IP-Adressen ist nach dem obigen Schema auf ca. vier Milliarden begrenzt, wobei durch technische und administrative[9] Einschränkungen nicht alle Adressen genutzt werden können. Aus diesen Gründen, sowie durch die Notwendigkeit der Anbindung von weiteren Netzwerken, wurden Subnetze eingeführt. Jedes Subnetz benötigt nur eine weltweit eindeutige IP-Adresse, auf welche sämtliche im Subnetz vorhandenen Computer abgebildet werden.

2.2.3 Dienste

Über eine große Anzahl von Computern im Internet werden eine Vielzahl von Diensten angeboten. Hier eine Auflistung der Wichtigsten:

- **WWW** - Das World Wide Web basiert auf dem HTTP-Protokoll [28], welches die Übermittelung von untereinander verknüpften Webseiten erlaubt. Eine Webseite kann beliebige visuelle und akustische Informationen enthalten, wie Texte, Bilder, Musik und Videos.

- **E-Mail** - Über verschiedene Protokolle [13], [55], [10], [37] können beliebige Nachrichten von einem Rechner zum einem anderen verschickt werden.

- **FTP** - Das File Transport Protocol dient dem Austausch von Dateien zwischen verschiedenen Rechnern [26].

- **Telnet und SSH** - Telnet [25] sowie der sicherere Nachfolger SSH dienen dem Zugriff auf entfernt stehende Rechner.

- **Tauschbörsen** - Tauschbörsen wie Napster oder Gnutella erlauben es den Benutzern, beliebige Dateien frei miteinander zu tauschen.

2.3 IT-Sicherheit

Zum Thema IT-Sicherheit hat Bruce Schneier ein Grundlagenwerk verfasst [39]. Allgemeinere Betrachtungen befinden sich in [40]. Die folgende Beschreibung von IT-Sicherheit orientiert sich an dem Skript *Sicherheit in Rechnernetzen* von Andreas Pfitzmann [34].

[8] DNS steht für Domain Name Service.

[9] So besitzt z.B. die USA als Ursprungsland des Internet über die Hälfte aller verfügbaren Adressen, Länder wie Indien oder China, mit Milliarden Einwohnern, jedoch nur sehr wenige.

2.3.1 Was ist IT-Sicherheit?

IT-Sicherheit befasst sich mit der Absicherung von Computern und der dazugehörigen Infrastruktur nach bestimmten Vorgaben (Schutzzielen). IT-Sicherheit umfasst einzelne Computer ebenso wie hochkomplexe Netzwerke von Computern. Eine umfassende Absicherung muss die Hardware-, Software- sowie Prozessebene umfassen.

Mehrseitige Sicherheit

Aktueller Stand der Forschung in verteilten Systemen ist die mehrseitige Sicherheit. Pfitzmann definiert mehrseitige Sicherheit wie folgt [48, S. 174]:

„Sicherheit für alle Beteiligten, wobei jeder anderen nur minimal zu vertrauen braucht.

- *Jeder hat individuelle Schutzziele.*
- *Jeder kann seine Schutzziele formulieren.*
- *Konflikte werden erkannt und Kompromisse ausgehandelt.*
- *Jeder kann seine Schutzziele im Rahmen des ausgehandelten Kompromisses durchsetzen.“*

Sicherheitspolitik

Die Sicherheitspolitik beschreibt die ausgehandelten Schutzziele in Form von Regeln und Attributen.

Sicherheitsanalyse

Die Sicherheitsanalyse eines gegebenen Szenarios umfasst mindestens die Punkte Schutzziele, mögliche Angreifer sowie den Entwurf hinsichtlich der Sicherheit. Das Thema Schutzziele wird im folgenden Abschnitt behandelt. Bei der Aufstellung der möglichen Angreifer muss geklärt werden, wer als Angreifer gegen die Schutzziele welcher Parteien auftreten kann. Danach müssen die Interessen, Stärken und Fähigkeiten möglicher Angreifer analysiert werden. Basierend auf den Schutzzielen und möglichen Angreifern kann dann ein Sicherheitsentwurf erfolgen. Dieser sollte geschützte Bereiche, Zugangs- und Zugriffskontrollmaßnahmen definieren. Weiterhin werden technische Möglichkeiten zur Durchsetzung von Schutzzielen festgelegt.

Zusammenfassend ist festzustellen, dass IT-Sicherheit aufgrund der hohen Komplexität sowie Dynamik der dahinter stehenden Menschen, Computer und Infrastruktur kein Produkt, sondern ein Prozess ist, der ständig angepaßt und erweitert werden muss.

2.3.2 Schutzziele

Eine Sicherheitsanalyse beginnt mit der Feststellung der einzelnen Schutzziele der an der Kommunikation beteiligten Personen. Dazu wird ermittelt, welche Personen, in welchen Rollen, mit dem System zu tun haben werden. Für jede Rolle, bzw. jede

Person müssen individuelle Schutzziele festgelegt werden. Da sich einige der Schutzziele zwischen verschiedenen Parteien widersprechen können, müssen Kompromisse ausgehandelt und festgelegt werden.

Die drei wichtigsten Schutzziele können unter dem englischen Kürzel CIA zusammengefasst werden. Dieses steht für:

- **Confidentiality – Vertraulichkeit**. Die Vertraulichkeit sichert die Geheimhaltung von Daten. Niemand außer den Kommunikationspartnern kann den Inhalt der Kommunikation erkennen.

- **Integrity – Integrität**. Die Integrität versichert die Unverfälschtheit der Kommunikation.

- **Availability – Verfügbarkeit**. Die Verfügbarkeit sichert die Nutzbarkeit von Ressourcen und Diensten, wenn ein Teilnehmer diese benutzen möchte.

Weitere Schutzziele nach [48, S. 175] werden im Folgenden erläutert. Diese haben verschiedene Wechselwirkungen untereinander[48, S. 179].

- **Verdecktheit**. Die Verdecktheit versteckt die Existenz der Übertragung von vertraulichen Daten.

- **Anonymität**. Die Anonymität sichert die Nutzbarkeit von Ressourcen und Diensten, ohne dass der Benutzer seine Identität preisgibt.

- **Unbeobachtbarkeit** Die Unbeobachtbarkeit sichert die Nutzbarkeit von Ressourcen und Diensten, ohne das dies andere beobachten können.

- **Zurechenbarkeit** Die Zurechenbarkeit sichert, dass das Senden oder Empfangen von Informationen gegenüber Dritten bewiesen werden kann.

- **Erreichbarkeit** Die Erreichbarkeit sichert, dass eine Ressource oder ein Dienst kontaktiert werden kann, sobald dies erwünscht ist.

- **Verbindlichkeit** Die Verbindlichkeit sichert, dass ein Benutzer belangt werden kann, seine Zusagen innerhalb einer bestimmten Frist einzulösen.

- **Pseudonymität** Die Pseudonymität sichert die Nutzbarkeit von Ressourcen und Diensten, ohne dass der Benutzer seine Identität preisgibt, ihm aber trotzdem die Nutzung zurechenbar ist.

Technische Möglichkeiten zur Durchsetzung von Schutzzielen

Im Folgenden werden technische Möglichkeiten zur Durchsetzung verschiedener Schutzziele aufgezeigt. Neben den technischen Möglichkeiten gibt es jedoch auch juristische und gesellschaftliche Lösungen.

2.3.3 Protokolle

Bruce Schneier definiert ein Protokoll folgendermaßen [39, S. 25]:

„Ein Protokoll dient der Durchführung einer bestimmten Aufgabe und besteht aus einer Folge von Aktionen, an denen zwei oder mehr Parteien beteiligt sind.“

Weitere Bedingungen für ein Protokoll sind, dass alle Beteiligten das Protokoll kennen und sich an die Regeln halten müssen. Weiterhin muss ein Protokoll eindeutig und vollständig sein. Protokolle formalisieren die Vorgehensweise und abstrahieren damit von einer konkreten Implementation.

Es gibt Protokolle mit Vermittlern, Schiedsrichtern oder eigenständige Protokolle. Eigenständige Protokolle benötigen nur die direkt an der Aktion beteiligten Parteien. Ein Protokoll mit Vermittlern benötigt eine unabhängige dritte Partei, welche den kritischen Vorgang durchführt. Ein Protokoll mit Schiedsrichtern benötigt eine dritte Partei nur bei Problemen. Dies ist vergleichbar mit einem Richter. Jede Partei legt ihre Beweise vor und der Schiedsrichter entscheidet danach.

Wenn es für ein Problem bereits ein bestehendes, gut erprobtes, Verfahren ohne bekannte Mängel gibt, so sollte dies auch zur Problemlösung verwendet werden. Geheime Protokolle und deren Implementierung (Security by Obscurity) wiesen in der Vergangenheit oftmals schwerwiegende Fehler auf, welche durch eine öffentliche Entwicklung und Diskussion wahrscheinlich hätten verhindert werden können.

2.3.4 Kryptographie

Die Kryptographie befasst sich mit der Absicherung von Nachrichten. Dazu wird eine im Klartext vorliegende Nachricht in einen Chiffretext überführt. Der Chiffretext kann nur in Zusammenhang mit einem Geheimnis (Dechiffrierschlüssel) wieder in den Klartext überführt werden. Die verwendete Methodik zur Ver- und Entschlüsselung wird zumeist als mathematische Funktion dargestellt.

In den Anfängen der Kryptographie war die verwendete Funktion gleichzeitig das Geheimnis, z.B. bei der Cäsar-Chiffrierung[10]. Diese Vorgehensweise ist heute aus vielfältigen Gründen überholt [39, S. 3].

Die moderne Kryptographie verwendet Schlüssel als Geheimnisträger. Diese werden je nach Algorithmenklasse (siehe nachfolgende Beschreibung) in die Verschlüsselungsfunktion eingesetzt. Die Sicherheit basiert ausschließlich auf den Schlüsseln, so dass die verwendeten Algorithmen öffentlich überprüft und analysiert werden können.

Im Folgenden werden verschiedenen Algorithmenklassen für kryptographische Funktionen vorgestellt. Diese können dann wieder in verschiedene Unterklassen bezüglich ihrer Sicherheit eingeteilt werden [34, S. 47].

Symmetrische Kryptographie

Bei der symmetrischen Kryptographie lässt sich der Dechiffrierschlüssel aus dem Chiffrierschüssel ableiten. Meist sind beide aus Gründen der Einfachheit identisch. Sender und Empfänger müssen vor der Übermittlung von vertraulichen Informationen einen Schlüssel austauschen (Key). Da sich Chiffrier- und Dechiffrierschlüssel

[10] Cäsar substituierte jeden Buchstaben des Klartextes durch den drei Buchstaben später folgenden. Die Cäsar-Chiffrierung ist z.B. hier beschrieben: `http://www.wikipedia.org/wiki/Caesar_cipher`

voneinander ableiten lassen oder identisch sind, werden sie gleichgesetzt. Damit kann der symmetrische Algorithmus wie folgt definiert werden:

$$Encrypt_{Key}(Plaintext) = Chiphertext \tag{2.1}$$
$$Decrypt_{Key}(Chiphertext) = Plaintext \tag{2.2}$$
$$Decrypt_{Key}(Encrypt_{Key}(Plaintext)) = Plaintext \tag{2.3}$$

Verbreitete Standards für symmetrische Kryptographie sind der Data Encryption Standard (DES) [30] sowie der neuere Advanced Encryption Standard (AES) [29]. In dieser Arbeit wird eine Erweiterung von DES verwendet, welche als 3DES bekannt ist[11]. Diese führt mehrere DES Operationen nacheinander, aber mit unterschiedlichen Schlüsseln durch und erhöht damit die effektive Schlüssellänge um den Faktor zwei oder drei.

Asymmetrische Kryptographie

Bei der asymmetrischen Kryptographie (auch als Algorithmen mit öffentlichen Schlüsseln oder Public-Key-Kryptographie bezeichnet), sind Chiffrier- und Dechiffrierschlüssel verschieden. Der Dechiffrierschlüssel sollte sich nicht aus dem Chiffrierschlüssel berechnen lassen[12]. Der Chiffrierschlüssel (Key_{Pub}) einer Person `P` kann somit veröffentlicht werden. Jeder der `P` eine vertrauliche Nachricht schicken möchte, kann diese mit Key_{Pub} verschlüsseln. Entschlüsseln kann sie nur `P`, da nur dieser den entsprechenden Dechiffrierschlüssel (Key_{Priv}) zu Key_{Pub} besitzt. Der Schlüsselaustausch zwischen Kommunikationspartnern wird somit wesentlich vereinfacht, da der Chiffrierschlüssel nicht geheimgehalten werden muss. Der Chiffrierschlüssel wird auch öffentlicher Schlüssel und der Dechiffrierschlüssel auch privater Schlüssel genannt. Der asymmetrische Algorithmus kann wie folgt definiert werden:

$$Encrypt_{Key_{Pub}}(Plaintext) = Chiphertext \tag{2.4}$$
$$Decrypt_{Key_{Priv}}(Chiphertext) = Plaintext \tag{2.5}$$
$$Decrypt_{Key_{Priv}}(Encrypt_{Key_{Pub}}(Plaintext)) = Plaintext \tag{2.6}$$

Ein weit verbreiteter asymmetrischer Algorithmus ist RSA [45].

Hybride Kryptographie

Trotz der Vorteile der asymmetrischen Kryptographie, wie dem vereinfachten Schlüsselaustausch, kann diese nicht überall eingesetzt werden. Grund dafür ist die hohe Komplexität der benötigten zugrunde liegenden mathematischen Operationen. So ist die symmetrische Kryptographie um den Faktor 100-1000 schneller als die asymmetrische Kryptographie.

Die hybride Kryptographie versucht nun die Vorteile der symmetrischen Kryptographie (Geschwindigkeit) und der asymmetrischen Kryptographie (einfacher Schlüsseltausch) zu vereinen. Vom Prinzip her handelt es sich um ein Protokoll: Kommunikationspartner `A` sowie `B` besitzen bereits den öffentlichen Schlüssel des jeweils

[11] siehe `http://www.wikipedia.org/wiki/Triple_DES`

[12] Leider ist kein solcher Algorithmus bekannt, jedoch kann die benötigte Zeit zum Berechnen des Dechiffrier- aus dem Chiffrierschlüssel astronomisch groß sein.

anderen ($KeyA_{Pub}$, $KeyB_{Pub}$). `A` generiert daraufhin einen symmetrischen Schlüssel (Key) und chiffriert diesen mit dem öffentlichen Schlüssel von `B`. Nur dieser kann Key aus der Nachricht dechiffrieren. `A` und `B` verwenden daraufhin den symmetrischen Schlüssel Key für ihre Kommunikation, welcher auf einem sicheren Wege ausgetauscht wurde:

1. $A \Rightarrow B : Ciphertext = Encrypt_{KeyB_{Pub}}(Key)$
2. $B : Key = Decrypt_{KeyB_{Priv}}(Chipertext)$

Dieses Protokoll ist sehr einfach gehalten, so dass eine Reihe von Fragen auftauchen, welche jedoch durch erweiterte Protokolle gelöst werden können:

- Wie kann sich `A` sicher sein, wirklich den öffentlichen Schlüssel von `B` zu besitzen?
- Wie kann `B` sicher sein, dass die Nachricht wirklich von `A` stammt?
- Wie können Wiederholungsangriffe verhindert werden (d.h. jemand schickt eine alte Anfrage von `A` erneut an `B`, obwohl der darin enthaltene Schlüssel Key bereits bekannt ist)?

Zur Beantwortung dieser Fragen können die im folgenden beschriebenen Techniken eingesetzt werden. Weiterführende Antworten finden sich u.a. in [39].

2.3.5 Digitale Signaturen

Ein weiteres wichtiges Werkzeug zur Durchsetzung von Schutzzielen ist die digitale Signatur (=digitale Unterschrift). Die digitale Signatur berechnet aus einem Klartext einen Chiffretext, welcher an den Klartext angehangen wird. Der Klartext ist für jedermann lesbar. Wird er jedoch geändert, so stimmt der anhängende Chiffretext nicht mehr überein. Zur Überprüfung auf die Unverfälschtheit kann der Chiffretext entschlüsselt werden und mit dem Klartext verglichen werden.

Laut [39, S. 41] gibt es fünf wichtige Merkmale für eine Unterschrift. Eine Unterschrift ist:

1. authentisch,
2. fälschungssicher,
3. sowie nicht wiederverwendbar.
4. Das unterzeichnete Dokument ist unveränderbar.
5. Die Unterschrift kann nicht zurückgenommen werden.

Diese Eigenschaften sind in der Realität schwierig durchzusetzen, können aber mittels symmetrischer oder asymmetrischer Kryptographie auf Computern durchgeführt werden. Die symmetrische Variante besitzt jedoch viele Nachteile, so kann z.B. auch der Überprüfer einer Unterschrift diese fälschen, da er den dafür nötigen Schlüssel besitzen muss.

Asymmetrische digitale Signaturen nutzen die Möglichkeit, die Funktion des öffentlichen und privaten Schlüssels zu tauschen. Der Unterzeichner eines Dokumentes verwendet seinen privaten Schlüssel um dieses zu chiffrieren und an den Klartext anzuhängen. Anschließend kann mit Hilfe des dazugehörigen öffentlichen Schlüssels der Chiffretext entschlüsselt werden und auf Gleichheit mit dem Klartext geprüft werden. Es gibt jedoch keine Möglichkeit mit Hilfe des öffentlichen Schlüssels den Chiffretext zu verändern. Eine asymmetrische digitale Signatur erfüllt somit alle fünf Merkmale einer Unterschrift, sofern nur der Unterzeichner im Besitz des privaten Schlüssels ist. Ein oft benutzter Standard für digitale Signaturen ist RSA.

Der Aufwand für diese Art der digitalen Signatur ist sehr groß. Die Textgröße verdoppelt sich im Minimum und die mathematischen Funktionen für die asymmetrische Kryptographie benötigen, spätestens bei einem Buch oder einer größeren Datei, eine sehr lange Zeit. In praktischen Implementierungen wird aus diesem Grund auch nur ein so genannter Hashwert signiert.

2.3.6 Einweg-Hashfunktionen

Eine Hashfunktion bildet einen Eingabewert auf einen (im Allgemeinen) kürzeren Ausgabewert ab:

$$y = H(x) \tag{2.7}$$

Eine Einwegfunktion ist eine Funktion, welche sich in eine Richtung sehr leicht, in die andere jedoch nur sehr schwer berechnen lässt:

$$Einfach : y = f(x) \tag{2.8}$$
$$Schwer : x = f^{-1}(y) \tag{2.9}$$

Die Kombination aus obigem ergibt eine Einweg-Hashfunktion. Diese bildet einen Eingabewert auf einen Ausgabewert ab, und zwar so, dass sich der Eingabewert aus dem Rückgabewert nicht (oder nur sehr aufwändig) berechnen lässt. Das Ergebnis wird als Hashwert bezeichnet.

Da Hashfunktionen im Allgemeinen einen Eingabewert auf einen kleineren Hashwert abbilden, können mehrere Eingabewerte den gleichen Hashwert erzeugen. Eine gute Hashfunktion muss dies möglichst verhindern, sie heißt dann kollisionsfreie Hashfunktion.

Eine beliebte Hash-Funktion ist MD5 [27], welche einen 128-Bit Hashwert erzeugt. Der Secure Hash Standard (SHS) [31] wurde für den Einsatz mit DES entwickelt. Der verwendete Algorithmus SHA-1 erzeugt einen 160-Bit Hashwert.

2.3.7 Public-Key-Zertifikat

Ein Public-Key-Zertifikat ist ein öffentlicher Schlüssel einer Person, welcher von einer vertrauenswürdigen dritten Person `T` digital signiert wurde. Es enthält weitere Angaben wie Gültigkeitsbegin und -ende.

Möchte `A` an `B` eine Nachricht schicken, so kann er das Zertifikat von `B` von einer beliebigen Stelle erhalten. `A` muss lediglich das Zertifikat von `T` auf einem sicheren

Weg erhalten haben, um die Gültigkeit eines von `T` unterzeichneten Zertifikates zu überprüfen. Somit kann `A` sicher sein, den echten öffentlichen Schlüssel von `B` zu besitzen, ohne diesen auf einem sicheren Weg ausgetauscht zu haben.

Sehr weit verbreitet sind X.509 Zertifikate [39, S. 652].

Kapitel 3

Vorhandene Technologien

In diesem Kapitel werden vorhandene Technologien vorgestellt, welche zur Erfüllung der Aufgabenstellung beitragen können. Viele der verwendeten Lösungen werden bereits im Campuskartenprojekt der TU-Berlin verwendet. Einige dieser Lösungen müssen zur Wahrung der Kompatibilität übernommen werden, bei anderen erfolgt jeweils eine kurze Abwägung der Vor- und Nachteile gegenüber alternativen Ansätzen.

Weiterhin werden bereits am Markt befindliche Multi–Applikations–Smartcard Lösungen betrachtet und eine Unterscheidung zum hier entworfenen System vorgenommen.

Abschließend wird kurz die TU–Campuskarte betrachtet. Ausführliche Erläuterungen, welche die Themen dieser Arbeit betreffen, befinden sich u.a. in [9], [36] sowie [35].

3.1 Java

Java bezeichnet eine plattformunabhängige Ausführungsumgebung und objektorientierte Programmiersprache der Firma Sun [2]. Java wurde ursprünglich Anfang der 90'er Jahre für kleine elektronische Geräte entwickelt. Durch die Integration in Web–Browser der Firma Netscape gelang die weltweite Verbreitung innerhalb kürzester Zeit. Heutzutage wird Java weniger in Web–Browsern oder auf Client-Systemen, sondern vielmehr im Server und Embedded–Bereich eingesetzt.

Zur Ausführung von Java–Programmen wird eine Virtual Machine benötigt, welche zu Bytecode kompilierte Java–Programme ausführen kann. Bytecode ist ein Art virtuelle Maschinensprache, welche prozessorunabhängig ist. Der Java–Bytecode wird zur Laufzeit durch die Virtual Machine in die Maschinensprache des jeweiligen Prozessors übersetzt und ausgeführt. Dies ermöglicht die genannte Plattformunbhängigkeit, sowie die Möglichkeit, Richtlinien zur Ausführung von Programmcode vorzugeben. Die Einhaltung dieser Richtlinien wird von der Virtual Machine während der Laufzeit überprüft. So ist es z.B. möglich, Programme im Web–Browser laufen zu lassen, welche weder auf die Festplatte des Benutzers zugreifen, noch Verbindungen zu anderen Servern aufbauen dürfen. Diese Richtlinien lassen sich für verschiedene Programme sehr fein granulieren. Der Ansatz, Programme zur Laufzeit zu überwachen, wird auch als Sandbox–Technologie bezeichnet. Java–Programme dürfen innerhalb ihrer Sandbox ohne Beschränkungen laufen, ein Ausbruch aus der Sandbox wird jedoch kontrolliert.

Als Programmiersprache betrachtet, ist Java eine nach modernen Kriterien entwickelte, stark typisierte, objektorientierte Sprache. Es gibt eine umfassende Klassenbibliothek sowie verschiedenste Erweiterungen in den Bereichen Sicherheit, Grafik oder Unternehmensanwendungen. Java verzichtet vollständig auf Zeiger und die damit verbundene Arithmetik, was einen sehr stabilen Code erzeugt. Buffer-Overflows und ähnliches sind im Rahmen eines Java–Programmes unbekannt. Weiterhin muss sich der Programmierer nicht selbst um die Speicherverwaltung kümmern, diese wird komplett von der Virtual Machine übernommen.

Der oben erwähnte Einsatz von Java–Technologie sowohl im Server als auch im Embedded–Bereich scheint auf den ersten Blick ein Widerspruch zu sein. Wie kann ein Programm einerseits auf einem großen Server laufen, andererseits sich aber mit sehr geringen Ressourcen eines Embedded–Gerätes zufrieden geben? Sicherlich werden dies nicht die gleichen Programme sein, aber die verwendete Technologie einer Virtual Machine schafft auf beiden Einsatzbereichen eine plattformunabhängige Lösung. Anwendungsentwickler sind nicht länger an einen Hersteller oder ein proprietäres Format gebunden, sondern können zwischen einer Vielzahl von Herstellern auswählen. Die einmal erstellten Programme sind ohne große Änderungen übertragbar. Weitere Vorteile, wie die Durchsetzung von Richtlinien zur Ausführung von Programmen oder der verbesserte Speicherschutz sind wesentliche Kriterien, welche sowohl auf Server, als auch auf kleinste Geräte zutreffen. Alle diese Vorteile erzeugen z.B. gerade im Chipkartenbereich einen wesentlichen Mehrwert. Java Umgebungen für Geräte mit stark begrenzter Kapazität erreichen selbstverständlich nicht den Funktionsumfang der Server–Versionen. Einige Einschränkungen für werden im folgenden Abschnitt gegeben, andere finden sich in [3].

Alternativen zu Marktdurchdringung und Technologie von Java gibt es sehr wenige. Die Firma Microsoft versucht mit der .NET Technologie einen ähnlichen Ansatz zu etablieren, dieser ist jedoch überwiegend auf Windows–Produkte beschränkt [1]. Im Rahmen der existierenden Campuskarten–Infrastruktur wurde mit großem Erfolg sehr stark auf die Java–Technologie zur Unterstützung heterogener Systeme gesetzt. Der Vorteil der Plattform- sowie Herstellerunabhängigkeit erwies sich an verschiedenen Stellen als sehr vorteilhaft. So konnte z.B. nach dem Rückzug eines großen Chipkartenproduzenten, relativ einfach auf einen anderen Hersteller umgestellt werden. Aufgrund der genannten Vorteile sowie Erfahrungen wird auch diese Arbeit in weiten Teilen auf der Java–Technologie basieren.

Zur Einführung in die Java–Programmierung gibt es eine große Anzahl von Büchern. Sehr umfassend sind [46] und [14]. Weiterführende Informationen befinden sich in [47].

3.2 Java Card

Ein weiteres Teilgebiet der Java–Technologie umfasst den Bereich der Chipkarten. Durch den Einsatz der Java Virtual Machine können einmal geschriebene Programme auf Chipkarten verschiedener Hersteller ausgeführt werden. Diese Technologie wurde von der Firma Sun Microsystems unter dem Namen Java Card entwickelt. Chipkarten, welche diese Technologie unterstützten, werden Java–Karten genannt. Die auf der Chipkarte ausgeführten Java–Programme werden in dieser Arbeit als Cardlets bezeichnet.

Eine Java Card Virtual Machine besitzt im Gegensatz zu leistungsfähigeren Versionen für Client- und Server–Systeme erhebliche Einschränkungen. Dies ergibt sich

aus der geringen Rechen– und Speicherkapazität einer Chipkarte. Es gibt zum Beispiel weder Fließkommaunterstützung, noch dynamisches Nachladen von Klassen zur Laufzeit. Ebenso fehlen Multi–Threading oder eine dynamische Speicherverwaltung. Ein wesentlicher Unterschied ergibt sich auch in der Lebenszeit von Objekten. Auf einer Java–Karte sind diese grundsätzlich persistent und existieren solange, bis sie explizit gelöscht werden. Die Trennung der Stromversorgung einer Java–Karte „friert“ den momentanen Objektzustand ein. Nach Wiederherstellung der Stromversorgung existieren die Objekte wie zuvor weiter. In einem klassischen Java–Programm werden die Objekte bei jedem Programmstart neu erzeugt und mit dem Beenden des Programmes gelöscht.

Ein weiterer Vorteil der Java–Technologie ist im Bereich der Multi–Applikations–Chipkarten durch die Sandbox–Ausführung von Programmen gegeben. Bei einer korrekten Implementierung der Virtual Machine können unterschiedliche Java–Programme auf einer Chipkarte unter keinen Umständen auf Speicher außerhalb ihres erlaubten Bereiches zugreifen. Dies ist für sicherheitsrelevante Anwendungen unabdingbar.

Zum Verständnis dieser Arbeit werden im Folgenden kurz einige zentrale Begriffe aus dem Chipkartenbereich erläutert. Technisch basieren Java–Karten auf der ISO/IEC Norm 7816. Nach dem einlegen in ein Terminal sendet die Chipkarte einen Answer to Reset (ATR). Dieser enthält verschiedene Parameter zum Aufbau der Verbindung. Eine wesentliche Unterscheidung findet zwischen dem T=0 sowie dem T=1 Übertragungsprotokoll statt. T=0 Chipkarten kommunizieren byteorientiert, während T=1 Chipkarten blockorientiert Daten übertragen. Der jeweilige Modus muss vom Terminal und den dazugehörigen Treibern unterstützt werden.

Die eigentliche anwendungsbezogene Kommunikation erfolgt immer durch eine Anfrage des Terminals an die Chipkarte, wobei diese eine Kommando–APDU (engl. Command APDU) als Eingabe erhält und eine Antwort–APDU (engl. Response APDU) zurückliefert. APDU ist die Abkürzung für Application Protocol Data Unit, welche Dateneinheiten der Anwendungsschicht bezeichnet. Diese sind transportprotokollunabhängig, so dass auf der APDU–Ebene theoretisch keine Unterscheidung zwischen T=0 und T=1 Chipkarten erforderlich ist. APDU's setzen sich immer aus einem Header und einem Body zusammen. Der Header enthält das eigentliche Kommando für die Chipkarte, im Body sind zusätzliche Informationen enthalten.

Als offene Plattformen besitzen Java–Karten eine große Marktdominanz. Weitere sind z.B. Basic–Card oder Multos. Da Java–Karten im Campuskartenprojekt vorgegeben sind, werden sie auch in dieser Arbeit verwendet.

Weiterführende Informationen zu Java–Karten befinden sich in [12].

3.3 Global Platform

Global Platform ist ein Zusammenschluss verschiedenster Branchen zur Standardisierung von Multi–Applikations–Smartcard Infrastrukturen. Einen Wegbereiter dazu stellte der Visa Open Platform Standard dar. Dieser wurde später in verschiedene Global Platform Standards übernommen. Zur Anwendung in dieser Arbeit kommen dabei der Card Production Guide, welcher den Prozess der Kartenproduktion beschreibt, sowie die Card Specification, welche die Schnittstellen einer kompatiblen Multi–Applikations–Chipkarte definiert. Aktuelle Chipkarten unterstützen lediglich ältere Versionen dieser Spezifikationen, welche noch unter dem Namen Visa Open

Platform veröffentlicht wurden. Auf diese wird im Folgenden Bezug genommen, auch wenn sie teilweise zu Global Platform umbenannt wurden.

3.3.1 Visa Open Platform Card Specification

Die von aktuellen Chipkarten unterstützte Version der Visa Open Platform Card Specification ist 2.0.1 [19], welche in dieser Arbeit kurz als VOP 2.0.1 bezeichnet wird.

Sämtliche im Rahmen dieser Arbeit eingesetzten Java–Karten sind weitestgehend VOP 2.0.1 kompatibel. Mittels der Java Card Technologie wird definiert, wie und in welchem Format Anwendungen auf der Karte ausgeführt werden. Eine Schnittstelle zur Außenwelt wurde nicht beschrieben. Diese ist im Card Specification Standard enthalten.

VOP 2.0.1 definiert drei verschiedene Klassen von Einträgen auf der Chipkarte, welche jeweils mit einem kompletten Lebenszyklus beschrieben werden.

- Die erste Klasse umfasst so genannte Security–Domains, welche alle globalen sicherheitsrelevanten Funktionen der Chipkarte umfassen. Dies bezieht sich im Besonderen auf die Bereitstellung eines sicheren Kanals zur Kommunikation, dem Sperren von einzelnen Anwendungen sowie der Verwaltung globaler PIN's. Eine besondere Security Domain ist der Card–Manager, welcher zusätzlich die Selektierung verfügbarer Anwendungen vornimmt und Kommandos an diese weiterleitet. Zudem verwaltet der Card–Manager, als standardmäßige Instanz auf jeder Chipkarte, sämtliche Lade-, Installations- und Löschvorgänge. Diese Funktionalität kann er an eine andere Security–Domain abgeben, welche sodann eine Security–Domain with Delegated Management genannt wird. Diese ist insofern wichtig, als dass mit ihrer Hilfe zusätzliche Möglichkeiten zur Sicherung der Verbindlichkeit der Kommunikation sichergestellt werden können.

- Die zweite Klasse stellen Applications dar. Diese sind initialisierte Programme auf der Chipkarte. Sie können von externen Anwendungen ausgewählt (selektiert) und verwendet werden.

- Die dritte Klasse stellen Loadfiles dar. Dies sind auf die Chipkarte geladene Dateien, die entweder von Applications benutzt werden oder zu einer Application installiert werden können.

Der VOP 2.0.1 Standard definiert weiterhin verschiedene APDU's, die zum Management einer Multi–Applikations–Chipkarte verwendet werden können. Der Standard umfasst unter anderem Kommandos zum Aufspielen von Loadfiles, dem Installieren von Applications, oder dem Abfragen von Status–Informationen. Standardmäßig werden alle diese Kommandos vom Card–Manager verarbeitet. Viele Hersteller haben proprietäre Erweiterungen vorgenommen, die nicht in VOP 2.0.1 spezifiziert wurden. Ein Beispiel ist die Abfrage des verfügbaren Kartenspeichers.

Weitere Abschnitte der Spezifikation enthalten Beschreibungen zur Bereitstellung eines sicheren Kanals zur Kommunikation mit einer Security–Domain. Dabei wird die Signatur und Verschlüsselung von APDU's definiert.

3.3.2 Visa Open Platform Card Production Guide

Der Visa Open Platform Card Production Guide in der Version 2.02 [56] beschreibt Prozesse zur Produktion von Multi–Applikations–Chipkarten mit Hilfe der Open Platform. Relevant im Rahmen dieser Arbeit sind die Ableitungen verschiedener Schlüssel für Security-Domains einer Chipkarte. Diese werden zur gegenseitigen Authentisierung mit dem Card–Manager benötigt, der wiederum zur Rekonfiguration der Chipkarten verwendet wird.

Ferner werden so genannte Card Production Life Cycle (CPLC) Daten definiert, welche sich jederzeit auslesen lassen. Mit Hilfe dieser Daten kann eine eindeutige Kartennummer (CUID) zu einer Chipkarte gebildet werden. Dies erfolgt nach folgendem Schema:

$$CUID = IC_{fabricator} + IC_{type} + IC_{batch} + IC_{serial} \tag{3.1}$$

Die Elemente IC_x sind jeweils Byte–Arrays, die aus den CPLC–Daten entnommen wurden. CUID ist eine Konkatenation dieser Werte.

3.4 Opencard Framework

Das Opencard Framework (OCF) beschreibt eine plattformunabhängige Anbindung von Chipkartenlesern mittels der Java–Technologie [4]. Dazu definiert das Framework je eine Schnittstelle zur Anwendungs– und zur Kartenleserseite. Der Hersteller eines Chipkartenterminals kann einen nativen Java–Treiber für das OCF bereitstellen, über welchen die Anwendung mit der Chipkarte kommunizieren kann. Als alternative Anbindung existieren verschiedene Wrapper, welche z.B. PC/SC–Treiber auf das OCF abbilden. PC/SC ist ein Standard von Microsoft zur Integration von Chipkartenlesern in das Windows–Betriebssystem.

Weiterhin definiert der Standard sogenannte CardServices, die das Gegenstück zu einer Anwendung auf der Chipkarte darstellen sollen. Sobald ein neues Programm auf die Chipkarte aufgebracht und ein dazugehöriger CardService bereitgestellt wurde, kann über diesen auf die Anwendung auf der Chipkarte zugegriffen werden. Um technische Details wie APDU's muss sich die Anwendung nicht kümmern.

Das Opencard Framework wird im Rahmen des Campuskartenprojektes zur Anbindung von Chipkartenlesern verwendet. Dies liegt darin begründet, dass das OCF der einzige unter Java verfügbare Standard zur Anbindung von Kartenlesern ist. Im Rahmen dieser Arbeit wird gezeigt werden, dass die Anbindung von Anwendungen über spezielle CardServices des OCF viele Nachteile mit sich bringen kann.

3.5 Web–Browser– und Serversicherheit

Das Thema Web–Browser– und Serversicherheit ist äußerst vielfältig. Im Folgenden wird daher nur ein kleiner Ausschnitt betrachtet, welcher bei der Konfiguration des Multi–Applikations–Frameworks für Chipkarten unbedingt beachtet werden sollte.

3.5.1 HTTPS

Eine Erweiterung des HTTP–Protokolls ist HTTPS, welches eine sichere Kommunikation zwischen Web–Browser und Web-Server bereitstellt. HTTPS verwendet zur

sicheren Kommunikation entweder das von der Firma Netscape entwickelte Secure Socket Layer Protokoll (SSL) oder den Nachfolger Transport Layer Security (TLS) [11]. HTTPS über TLS ist in [38] beschrieben.

Zusätzlich zur Sicherung der übertragenen TCP/IP–Pakete mittels SSL/TLS muss der Web–Browser die Identität des Web–Servers mittels eines Public–Key–Zertifikates überprüfen. Dieses ist von einer vertrauenswürdigen Stelle unterzeichnet und enthält den Namen des Web–Server. Somit wird verhindert, dass sich Angreifer zwischen die Kommunikation klinken können (Man in the Middle Attack).

3.5.2 Firewalls

Firewalls für IT-Systeme dienen, wie ihr Vorbild aus der realen Welt, dem Schutz der Infrastruktur. Sie können grob in die drei Kategorien Paketfilter, Application–Level–Firewalls sowie Personal Firewalls klassifiziert werden.

Zur Sicherung des Multi–Applikations–Frameworks für Chipkarten bieten sich Paketfilter–Firewalls an. Diese arbeiten auf TCP/IP–Ebene und können Zugriffe auf bestimmte IP–Adressen und Ports begrenzen oder sperren. Üblicherweise werden Paketfilter–Firewalls an Verbindungen zwischen verschiedenen Subnetzen eingesetzt. So sollte ein Web–Server z.B. außerhalb seines Subnetzes nur über die zum Zugriff benötigten HTTP– oder HTTPS–Ports erreichbar sein. Alle anderen Zugriffe werden durch die Firewall verworfen.

3.6 Web–Server

Ein Web–Server stellt HTML–Webseiten für Web–Browser zur Verfügung. Die Webseiten können dabei statisch vorliegen oder durch Programme generiert werden. Komplexere Programme erzeugen Webseiten aus verschiedenen Hintergrunddiensten, wie z.B. Datenbanken. Weit verbreitete Technologien dafür sind CGI oder Skripte, welche ihrerseits auf statischen Webseiten basieren, wie PHP.

Zur Java–Technologie gehören verschiedene Techniken zur Erzeugung von dynamischen Webseiten. Zur Generierung komplexer, dynamischer Webseiten dienen Servlets, die nichts anderes als ein Java–Programm mit einer definierten Schnittstelle zur Ein- und Ausgabe von Webseiten sind. Servlets besitzen alle Vorteile von Java. So können u.a. alle existierenden Java–Klassen integriert werden. Zusätzlich laufen Java–Programme im Web–Server oftmals schneller als CGI–Programme, da sie nicht jedes Mal neu erzeugt und gestartet werden müssen. Alle Servlets werden von einer Virtual Machine verwaltet.

Eine freie Implementierung eines Servlet Web–Servers ist Tomcat [5]. Dieser Web–Server wird im Rahmen des Campuskartenprojektes für verschiedene Zwecke eingesetzt und ist sehr ausgereift. Tomcat ist ein nicht–kommerzielles Produkt, welches frei verwendet werden darf und im Quelltext erhältlich ist. Für höhere Anforderungen stehen sehr viele kommerzielle Servlet Web–Werver von renommierten Anbieter zur Verfügung.

3.7 Objektübertragung und -kommunikation

Zum Übertragen von Objekten über Netzwerke existieren verschiedene proprietäre und offene Standards. Zur Serialisierung von Objekten innerhalb des Java–Frame-

works können diese direkt binär oder in XML serialisiert werden. Eine Re-Serialisierung kann nur mit Hilfe der entsprechenden Java–Klassen stattfinden. Dieser Ansatz ist sehr gut zur einfachen Kommunikation zwischen verschiedenen verteilten Java–Programmen geeignet.

Mittels SOAP [59] lassen sich Objekte unabhängig von bestimmten Programmiersprachen über HTTP tunneln. Alle Attribute des Objektes werden dabei in eine XML-Darstellung transformiert und in eine Webseite gepackt.

Für komplexere Anwendungen steht unter Java die Remote Method Invocation (RMI) zur Verfügung. Hierbei bieten verschiedene Server Objekte an, welche von einem Client wie lokal vorhanden verwendet werden können.

Als interoperable Lösung zur Objektkommunikation über Netzwerke wurde von der Object Management Group (OMG) die Common Object Request Broker Architecture (CORBA) [32] übernommen. Diese definiert eine vollständige Middleware zur plattformübergreifenden Objektkommunikation.

Im Rahmen dieser Arbeit wird die binäre Java–Serialisierung verwendet, da diese im Vergleich zu den anderen genannten Lösungen am performantesten ist.

3.8 Multi–Applikations–Chipkarten Lösungen

Wesentliche Vorgaben für Multi–Applikations–Chipkarten Lösungen wurden durch die Global Platform Spezifikationen definiert. Diese beziehen sich im Besonderen auf den Prozess der Sicherung von der Herstellung der Chips über die Integration in eine Trägerkarte bis hin zur Lieferung an die kartenausgebende Instanz. Für eine komplette Infrastruktur sind allerdings weitere Komponenten nötig, von denen einige in dieser Arbeit erläutert werden.

Eine mögliche Multi–Applikations–Chipkarten Lösung hat die Firma Bell ID mit ihrem ANDiS Card & Application Management System auf ihrer Webseite (`http://www.bellid.com`) vorgestellt. Im Whitepaper zu diesem System kommt so ziemlich jedes Schlagwort, welches mit Chipkarten in Verbindung gebracht werden kann, vor. Es werden viele benötigte Komponenten beschrieben und als optimal angepriesen. Besonders hebt Bell ID die Kompatibilität ihrer Software mit allem, was es auf dem Chipkarten–Markt gibt, hervor. Einen Einblick in diese „Kompatibilität“ ergibt sich bereits dadurch, dass ein Interessent die Webseite von Bell ID nur mit dem Web–Browser einer marktbeherrschenden Firma einsehen kann. Eine Demonstration oder weitere technische Unterlagen sind auf der Webseite nicht verfügbar.

Die in Deutschland weit verbreitete Geldkarte [54] ist ebenfalls eine Multi–Applikations–Chipkarte. Die Geldkarten–Funktionalität ist nur eine mögliche Anwendung des Chipkarten-Betriebssystems SECCOS (Secure Card Operating System). SECCOS unterstützt das Nachladen und Installieren von Programmcode nach der Ausgabe. Allerdings benötigt jeder Kartentyp unterschiedlichen Maschinencode, da die unterstützten Chipkarten auf verschiedener Hardware aufbauen. Dies stellt einen nicht unerheblichen Nachteil für die Rekonfiguration der Geldkarte dar. Trotzdem gibt es in verschiedenen deutschen Städten Projekte, wie z.B. in Bremen, wo elektronische Tickets für den öffentlichen Nahverkehr auf die Geldkarte aufgebracht wurden. Als wesentlicher Nachteil ist zu nennen, dass die Geldkarte kein offener Standard ist, sondern vom ZKA (Zentraler Kredit Ausschuss) spezifiziert und überwacht wird. Die aktuelle Spezifikation ist nur käuflich zu erwerben.

Den exemplarisch genannten Systemen ist gemeinsam, dass sie zwar mehr oder

weniger stark auf Standards basieren, aber dennoch nicht frei oder im Quelltext verfügbar sind. Zur Integration in ein vorhandenes, sicheres System ist jedoch eine genaue Analyse der zu übernehmenden Programmteile unabdingbar. In einem weiteren Schritt müssen Anpassungen vorgenommen werden, welche sich auf die zu integrierenden Programmteile beziehen. Dabei ist es meist eine ungünstige Vorgehensweise, das vorhandene System so anzupassen, dass es mit Fremdkomponenten arbeitet. Gerade in sicherheitskritischen Programmteilen dürfte dies sehr schwierig sein.

Diese Arbeit will jedoch anhand einer Analyse, eines Entwurfs und einer prototypischen Implementierung eine Bogen über alle benötigten Komponenten spannen, um daraus neue Anregungen für ein Multi–Applikations–Framework für Chipkarten zu entwickeln. Vorhandene, proprietäre oder geschlossene Systeme lassen sich dazu nur begrenzt verwenden.

3.9 TU–Campuskarte

Das Projekt „Campuskarte“ befaßt sich mit der Einführung einer multi–funktionalen Chipkarte für alle Hochschulangehörigen der TU-Berlin. Zu diesem Zweck wurde eine komplette Infrastruktur entworfen und implementiert. Die Chipkarte stellt als Hauptanwendung eine Authentisierungs-, Signatur- sowie Verschlüsselungsfunktionalität bereit. Bei der Authentisierung kommt ein neu entwickeltes Verfahren zum Einsatz, welches die Persönlichkeitsrechte des Benutzers besonders schützt [22].

Die TU–Campuskarte basiert auf einer Trägerkarte, die sowohl einen kontaktbehafteten Java–Karten Chip als auch einen kontaktlosen Mifare Chip enthält. Zum Zeitpunkt der Ausgabe der Campuskarte, im Wintersemester 2003/2004, befindet sich auf dem Java–Karten Chip lediglich die Campuskarten–Anwendung. Diese stellt die oben erwähnten Funktionalitäten zur Verfügung. Der Mifare Chip wurde mit einer elektronischen Geldbörse vorpersonalisiert. Auf beiden Chips ist noch ausreichend Speicherplatz für zukünftige Erweiterungen vorhanden.

In dieser Arbeit wird der kontaktbehaftete Java–Karten Chip betrachtet, da nur dieser die Ausführung eigener Programme erlaubt. Der Mifare Chip dient lediglich zur Datenspeicherung.

Weitere Informationen zur Campuskarte der TU-Berlin befinden sich in [9].

Kapitel 4

Anforderungsanalyse

Dieses Kapitel beschreibt die Anforderungen an das zu entwickelnde System. Es handelt sich bereits um eine Analyse der Anforderungen, mit dem Ziel weitere Anforderungen zu finden. Da sehr viele Prozesse in der Softwareentwicklung iterativ sind und somit ihren endgültigen Zustand erst nach mehreren Durchläufen erreichen, umfasst diese Anforderungsanalyse auch bereits Lösungsansätze. Dies begründet sich darin, dass diese wiederum als eigene (Teil)-Aufgaben aufgefasst werden können und auch einer Anforderungsanalyse bedürfen. Diese (Unter)-Anforderungen (wie auch die Systemanalyse und der Systementwurf in den folgenden Kapiteln) lagen zum Beginn der Arbeit natürlich nicht in dieser endgültigen Form vor. Aus Verständnisgründen werden jedoch nur die letzten Schritte der Iteration dargestellt.

4.1 Überblick

Im ersten Unterabschnitt werden die aus der Aufgabenstellung gegebenen Anforderungen erläutert sowie nicht funktionale Anforderungen definiert. Der im Folgenden dargestellte Systemüberblick sowie die wesentlichen Akteure ergaben sich erst in späteren Analyseschritten, sind aber für die nachfolgenden Sicherheits- sowie Systemanforderungen notwendig.

4.1.1 Anforderungsbeschreibung

„Das Ziel der Diplomarbeit soll eine Erweiterung des vorhandenen Campuskarten Frameworks um die Möglichkeit der dynamischen Rekonfiguration der Chipkarte über offene Netze sein.“ (zitiert aus der Aufgabenstellung)

Hierbei muss zuerst definiert werden, was die dynamische Rekonfiguration der Chipkarte umfassen soll. Dazu definiere ich im Rahmen dieser Arbeit eine Multi-Applikations-Chipkarte als Behälter für verschiedene Programme auf der Chipkarte, die mit entsprechenden Anwendungen kommunizieren können. Ein gegenläufiger Ansatz, der momentan auch von der Campuskarte verfolgt wird, besteht darin, auf der Chipkarte nur ein Programm mit einer Authentisierung des Benutzers laufen zu lassen. Diesen Dienst können beliebig viele andere Anwendungen nutzen, die dafür notwendigen Daten werden auf Serversystemen gehalten. Wenn jedoch eine ständige Online-Verfügbarkeit nicht gewährleistet sein kann, bzw. datenschutzrechtliche Gründe gegen das letztere Verfahren sprechen, kommen die Vorteile der hier verwendeten Definition zum Tragen. Am Beispiel der Erweiterung der Campuskarte

für sichere E–Mail wird deutlich, dass ein monolithisches Konzept nur schwer zu erweitern ist. Die sichere E–Mail benötigt entsprechende Zertifikate sowie passende Kryptographiefunktionalität auf der Chipkarte. Diese wurden in das Authentisierungsprogramm, das auf der Chipkarte läuft, integriert. Nach der Ausgabe ist es technisch unmöglich neue Funktionalität, wie geänderte Hash-Funktionen oder andere Zertifikattypen zu unterstützen. Diese könnten für zukünftige Anwendungen aber benötigt werden. Meines Erachtens sinnvoller ist eine Aufteilung verschiedener Funktionalitäten in verschiedene Programme auf der Chipkarte, die vom Karten-Halter individuell nach seinen Bedürfnissen aufgespielt und gelöscht werden können.

Dieses dynamische Aufspielen und Löschen von Funktionalitäten auf der Chipkarte ist gleichzeitig der Kernpunkt dieser Arbeit. Da unter die Verwaltung des Chipkarteninhaltes durch den Karten-Halter noch andere Aspekte fallen, nenne ich es dynamische Rekonfiguration. Dem Karten-Halter soll es weiterhin möglich sein, seine Chipkarte möglichst überall und jederzeit rekonfigurieren zu können. Dazu bieten sich offene Netze an.

Eine Erweiterung des bestehenden Frameworks erwies sich aus vielfältigen Gründen als sehr schwierig. Einer dieser Gründe ist die mögliche Integration in ein flexibles Kartenmanagement, das in der vorhandenen Version des Frameworks nicht vorgesehen ist. Zur Unterstützung von Multi-Applikations-Chipkarten ist es nach meiner Auffassung aber unabdingbar. Weitere Punkte umfassen ein Applikations- sowie Rechtemanagement. Diese Anforderungen ergaben sich erst in späteren Iterationen der Analyse und Entwicklung. Weitere Betrachtungen werden in Kapitel 5.2 vorgenommen.

4.1.2 Nicht-funktionale Anforderungen

Die Dienste der vorhandenen Campuskarte können größtenteils über Web–Browser mit einem Chipkartenleser genutzt werden. Die bestehende Infrastruktur soll auch zur dynamischen Rekonfiguration von Chipkarten eingesetzt werden. Das bedeutet, der Karten-Halter soll zur Rekonfiguration der Campuskarte keine Konfigurationsänderungen oder Installationen an seinem System vornehmen müssen. Es gelten die gleichen Anforderungen wie für die Nutzung der Campuskarte [9]. Alle Systeme, welche durch das Campuskarten–Setup unterstützt werden, sollen auch in dieser Arbeit verwendet werden können.

Die server-seitige Software soll möglichst universell einsetz- sowie erweiterbar und auf verschiedenen Systemen lauffähig sein. Dazu bietet sich die Verwendung der Programmiersprache Java sowie des Web–Servers Tomcat an. Beide Produkte werden im Rahmen der existierenden Campuskarteninfrastruktur bereits verwendet, so dass für eine mögliche Integration kein größerer Aufwand notwendig ist.

Als Chipkarten sollen die bereits im Campuskartenprojekt verwendeten Java-Karten verwendet werden. Eine Erweiterung auf zukünftige Java-Karten sollte so einfach wie möglich sein.

Zusätzliche Komponenten wie Datenbanken oder Ähnliches sollen im Rahmen der prototypischen Implementierung nicht verwendet werden, sind aber für einen Produktionseinsatz eine sinnvolle Ergänzung.

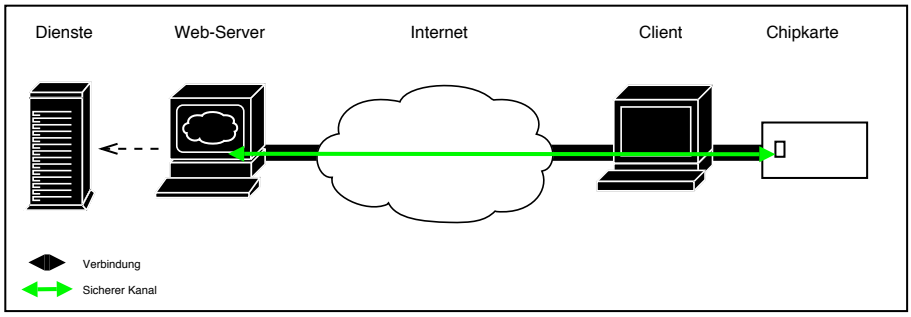

Abbildung 4.1: Grundlegender Systemaufbau

4.1.3 Systemaufbau

Im Laufe der mehrphasigen Anfordungs- sowie Systemanalyse kristallisierte sich ein Systemkonzept heraus, das entwickelt werden sollte. Um einen ersten Überblick zu geben, ist es bereits in der Anforderungsanalyse enthalten.

Der grundlegende Systemaufbau ist in Abbildung 4.1 dargestellt. Der Karten-Halter verwendet seine Chipkarte in einem Kartenleser, der an einen Client–Rechner angeschlossen ist. Dieser wiederum kommuniziert mit einem Web–Server, der über offene Netze, z.B. dem Internet, erreichbar ist. Zusätzlich wird die Verbindung über das Internet mit Standardprotokollen abgesichert. Für die Erbringung der eigentlichen Funktionalität sind verschiedene Dienste zuständig, auf welche der Web–Server zugreifen kann. Diese Dienste kommunizieren aus Sicherheitsgründen über eine gesicherte Verbindung mit der Chipkarte.

4.1.4 Wesentliche Akteure

Anhand der Aufgabenstellung sowie den vorhergehenden Erläuterungen ergeben sich bereits einzelne Akteure und ihre Aufgabenbereiche, die zum Verständnis der folgenden Unterkapitel in Abbildung 4.2 dargestellt sind.

Der Karten-Halter wurde bereits mehrmals erwähnt. Er ist im Kontext dieser Arbeit der Anwender, der seine Chipkarte rekonfigurieren möchte. Die dazu benötigten Dienste stellt ihm der Karten-Betreiber zur Verfügung. Neue Programme zum Aufspielen auf die Chipkarte bietet ein so genannter Cardlet-Anbieter an. Zur Kontrolle der Korrektheit und Unversehrtheit können diese Cardlets von einer dritten Instanz, dem Cardlet-Signierer, geprüft und digital unterschrieben worden sein. Nach einer erfolgreichen Installation oder dem Löschen eines Cardlets muss dies dem zugehörigen Anwendungs-Betreiber mitgeteilt werden, um weitere Aktionen wie Personalierung, Aktivierung oder Rechnungslegung durchführen zu können. Ein Anwendungs–Betreiber im Rahmen dieser Arbeit ist ein direktes Gegenstück zu einem Cardlet auf der Chipkarte. Er kommuniziert direkt mit der Chipkarte und stellt Funktionalitäten für andere Dienste bereit.

4.2 Sicherheitsanforderungen

Ein über offene Netze angebundenes System zur Rekonfiguration von Chipkarten muss über effiziente und effektive Sicherheitsmechanismen auf Seiten der Chipkarte und des Servers verfügen. Effizient deswegen, weil die Rechenleistung der Chipkarte

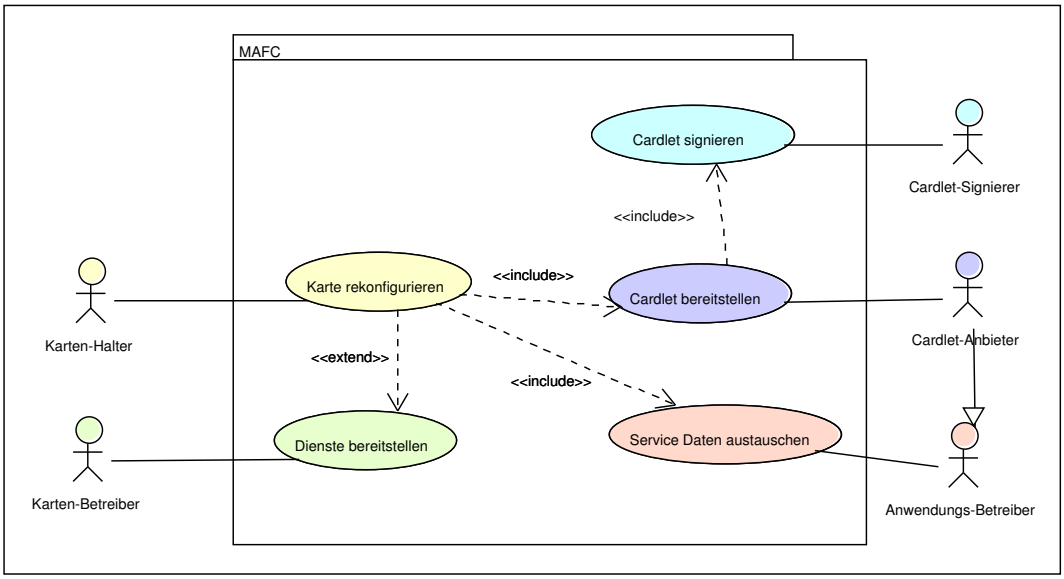

Abbildung 4.2: Wesentliche Anwendungsfälle

sehr gering und die Bandbreite der Verbindung zum Server im Allgemeinen nur sehr niedrig ist. Effektiv, um trotz dieser Randbedingungen ein sicheres Verfahren zu ermöglichen.

Dabei kann weder davon ausgegangen werden, dass das Kartenterminal und die Verbindung zum Rechner des Karten-Halters sicher sind, noch kann dies für den Rechner und die Internetanbindung desselben garantiert werden. Die einzigen sicheren Komponenten sind der Server sowie die Chipkarte selbst, wobei letztere sich nach der Ausgabe in einem physisch nicht kontrollierbarem Bereich befindet.

Sämtliche Kommunikation zwischen Chipkarte und Server muss daher vertraulich und authentisch möglich sein. Das Ausspionieren, Ändern, Unterbinden oder Wieder einspielen von kommunikationsbezogenen Daten muss möglichst verhindert werden.

Als zusätzliche Sicherheitsmaßnahme kann die Chipkarte, nach ihrer Ausgabe, nur von einer vertrauenswürdigen Stelle digital unterschriebene Cardlets akzeptieren. Damit ist gewährleistet, dass selbst bei einem Brechen der Verbindung keine unauthorisierten Cardlets auf die Karte gelangen.

Zur Erlangung eines sicheren Kommunikationskanals vom Server zur Chipkarte, muss eine gegenseitige Authentisierung zwischen beiden Teilen stattfinden. Der Klartext der übertragenen Daten darf nur auf dem Server sowie auf der Chipkarte vorliegen.

Die oben genannten Sicherheitsanforderungen werden in einer Sicherheitsanalyse betrachtet und sofern technisch möglich, auch im Systementwurf umgesetzt.

Nicht Aufgabe dieser Arbeit sind die Anbindung des Systems an ein Authentisierungs- sowie Rechtemanagementsystem. Ersteres ist sehr zielumgebungsabhängig und es gibt bereits sehr viele Lösungen auf diesem Gebiet, angefangen von einer einfachen Name/Passwort Abfrage bis hin zur pseudonymen Authentisierung der Campuskarte [22]. Sehr viel interessanter ist ein Rechtemanagement, d.h. wer darf was? Dies bezieht sich nicht nur auf externe Aktoren sondern auch auf systeminterne Zugriffe. Eine kurze Betrachtung findet sich auch in den folgenden Abschnitten, aufgrund der Komplexität des Themas kann eine Integration aber nicht Bestandteil dieser Arbeit sein.

4.3 Systemanforderungen

Der Abschnitt Systemanforderungen beschreibt weitere Anforderungen an das System, die nicht in der Aufgabenstellung beschrieben, jedoch im Laufe der Arbeit als notwendig erachtet wurden.

4.3.1 Dienste

Der Begriff Dienste bereitstellen taucht schon in Abbildung 4.2 als Anwendungsfall auf. In diesem Fall sind es chipkartenspezifische Dienste, die zur Rekonfiguration der Chipkarte benötigt werden. Sie kommunizieren direkt mit der Chipkarte. Weitere Dienste ergaben sich im Laufe der Analyse.

Chipkartendienste

Die Chipkartendienste sollen die Kommunikation mit der Chipkarte durchführen. Sie umfassen sowohl Low- als auch High-Level Funktionalität. Wie alle folgenden Dienste sollen sie so flexibel und erweiterbar wie möglich entworfen werden.

Kartenmanagement

Das Kartenmanagement beschreibt den kompletten Lebenszyklus der Chipkarte. Ansätze für einfache Chipkarten können aus [57] entnommen werden. Spezifische Prozesse für im Rahmen des Campuskartenprojektes eingesetzte Chipkartentypen sind auch in [19] sowie [56] enthalten. Eine Erweiterung für zukünftige Chipkarten ist in [20] beschrieben.

Das momentan im Rahmen des Campuskartenprojektes eingesetzte Karten-Status-Abfrage-System ist nur beschränkt verwendbar, da es sich entgegen seinem Namen nur auf den On-Card Teil einer Anwendung (Campuskarten–Cardlet) auf der Chipkarte bezieht. Ein Multi–Applikations–Chipkarten–Management muss aber den Status des Zustandes der gesamten Chipkarte umfassen. Die dabei erreichbaren Zustände unterscheiden sich von denen einer Anwendung.

Für die prototypische Implementierung wird nur ein einfacher Entscheider benötigt, der für eine gegebenen Chipkarte Rekonfigurationen erlaubt oder sperrt.

Cardletmanagement

Es wird ein Management zur Verwaltung neuer und vorhandener Cardlets für die Chipkarte benötigt. Diese Cardlets stellen den On-Card Teil einer Anwendung dar. Der Off-Card Teil kann z.B. ein Web–Server oder ein anderes Programm außerhalb der Chipkarte sein. Dieses Management muss Meta-Informationen für vorhandene Cardlets bereitstellen. Weiterhin müssen neue Cardlets zum Aufspielen zur Verfügung gestellt werden.

Rechtemanagement

Dieser bereits angesprochene Dienst ist ein wesentlicher Teil einer jeden sicheren IT-Infrastruktur. Es würde sich ein Dienst anbieten, der sowohl Subjekte (Personen, Rollen) als auch Objekte (andere Dienste) erkennt und Verhaltensregeln entsprechend auswertet. Die prototypische Implementierung soll aus Komplexitätsgründen

kein Rechtemanagement enthalten, so dass es in den folgenden Kapiteln nur am Rande betrachtet wird.

4.3.2 Benutzerschnittstelle

Da als nicht-funktionale Anforderung bereits die Nutzung eines vorhandenen Campuskarten kompatiblen Rechners vorgesehen ist, besteht die Schnittstelle des Karten–Halters zum System in einem Web–Browser, in dem ein Applet läuft. Der Web–Server stellt das Applet zum Download bereit und verwendet es anschließend zur Kommunikation mit dem Karten-Halter. In der prototypischen Implementierung soll die Konfiguration des Servers für alle anderen Aktoren über Konfigurationsdateien erfolgen.

4.3.3 Chipkarten

Die zu unterstützenden Chipkarten entsprechen denen der im Campuskartenprojekt eingesetzten. Die Unterstützung soll sich auf Visa Open Platform 2.0.1 [19] konforme Java-Karten beschränken. Trotzdem sollen bereits in Analyse und Entwurf Vorkehrungen zur Unterstützung von anderen Chipkartentypen getroffen werden.

Kapitel 5

Systemanalyse

Üblich ist, dass am Anfang der Analyse die Auswahl einer Methode steht, welche den Analyse-, Entwurfs- und eventuell sogar den Implementationsprozess umfasst. Im Rahmen der Lehrveranstaltung Softwaretechnik habe ich die Fusion-Methode kennen gelernt. Diese ermöglicht die objektorientierte Analyse und den nachfolgenden Entwurf eines kleinen bis mittelgroßen Softwaresystems. Der Analyse-Prozess nach Fusion beginnt mit der Erstellung eines Klassenmodells, in das alle gefundenen Objekte mit ihren Attributen und Beziehungen eingetragen werden. Je komplexer ein Softwaresystem, umso aufwendiger wird die Erstellung des Klassendiagramms, wenngleich es sich auch bei Fusion um einen iterativen Prozess handelt. Nach der Erstellung des Klassenmodells folgt die Generierung verschiedener Anwendungsfälle, in welchen systemexterne Aktoren und ihr funktionales Verhalten gegenüber dem System beschrieben werden. Weiterhin werden Zeitliniendiagramme zur genaueren Spezifikation von Anwendungsfällen verwendet. Der Analyse-Prozess nach Fusion mündet schließlich in einem Systemklassendiagramm, das systeminterne und -externe Klassen des Klassendiagramms trennt. Anschließend wird ein Operationsmodell für alle Systemoperationen aufgestellt. Beendet wird die Fusion-Analyse durch die Erstellung eines globalen Lebenszyklusmodells des gesamten Systems, das alle Systemoperationen umfassen muss. Weiter gehende Informationen befinden sich in [53].

Aus verschiedenen Gründen, die ich im Folgenden erläutere, habe ich auf die Verwendung von Fusion verzichtet. Alle nicht speziell gekennzeichneten Abbildungen entsprechen der UML Version 1.5 [33].

Mein Analyse-Prozess umfasst kein einziges Klassendiagramm, welches in Fusion die Grundlage für jeden weiteren Schritt bildet. Dies liegt darin begründet, dass es mir unmöglich war, alle zu verwendenden Objekte bereits in ihrer Gesamtheit am Anfang des Prozesses zu ermitteln. Vielmehr waren fast alle unbekannt. Der Fusion-Prozess basiert sehr stark auf der Abbildung von Objekten und Prozesses der realen Welt, welche bereits überwiegend bekannt sind. Zumindest habe ich Fusion nur an solchen Beispielen kennengelernt; auch die Dokumentation [53] orientiert sich sehr stark an realen Prozessen. Im Rahmen dieser Arbeit sind jedoch vor allem Anwendungsfälle vorgegeben, d.h.: „Was soll der Anwender machen?“, und nicht: „Wie soll er etwas tun?“. Sicherlich könnte man, basierend auf den gegebenen Anwendungsfällen, Abbildungen auf verschiedene Objekte finden und darauf basierend ein Klassendiagramm erzeugen. Eine weitere Bearbeitung ist im Rahmen einer Diplomarbeit mit praktischem Anteil jedoch viel zu umfangreich.

Um das Ziel der Diplomarbeit trotzdem zu erreichen, werde ich auf einer sehr

hohen und allgemeinen Abstraktionsebene beginnen und diese in den folgenden Kapiteln schrittweise verfeinern, um somit das System als Ganzes zu beschreiben.

Ich beginne daher die Systemanalyse mit der Erweiterung des bereits im Kapitel 4.1.4 aufgezeigten Anwendungsfalldiagramms. Dieses kann entworfen und erweitert werden, ohne dass spezielle Details wie Objekte, Attribute oder Kardinalitäten berücksichtigt werden müssen. Da ich, anders als im Fusion-Prozess, noch keine Objekte besitze, um Zeitliniendiagramme zur Verfeinerung von Anwendungsfällen aufzustellen, greife ich auf die Aktivitätsdiagramme der UML zurück. Diese erlauben mir Prozesse auf einem hohen Niveau zu beschreiben, können gleichzeitig aber auch bereits bekannte Details enthalten.

In diesem Kapitel werde ich neben der bereits beschriebenen Analyse auch auf Einschränkungen der Anforderungen sowie auf den vorhandenen Quelltext eingehen. Da sich Sicherheit als zentraler Faden durch diese Arbeit zieht, können aufgrund der Anwendungsfälle bereits alle relevanten Rollen erkannt und eine Sicherheitsanalyse durchgeführt werden.

5.1 Einschränkungen

Den Inhalt der folgenden Kapitel möchte ich in einigen Bereichen einschränken. Dies betrifft die Beschreibung nicht implementierter Anforderungen. Diese umfassen das komplette Rechtemanagement, die Überprüfung digitaler Signaturen von Cardlets sowie ein erweitertes Karten–Status–Abfragesystem. Aus Gründen der Vereinfachung sind diese Anforderungen nicht berücksichtigt worden. Im Mittelpunkt steht die prototypische Implementierung, d.h. es wurde ein wesentliches Augenmerk auf die Erfüllung der primären Aufgabe des Karten-Halters im Rahmen dieser Arbeit, der Rekonfiguration der Chipkarte, gelegt. Viele sehr interessante Hintergrundsysteme wurden nur soweit analysiert, entworfen und implementiert, wie dies zur Erfüllung der primären Aufgabe notwendig ist. Trotzdem glaube ich, diese so flexibel und erweiterbar wie möglich gestaltet zu haben.

5.2 Vorhandener Quelltext

Eine in der Aufgabenstellung genannte Anforderung war die Analyse und mögliche Integration von vorhandenem Quelltext. Der für die Zwecke dieser Arbeit möglicherweise verwendbare, vorhandene Quelltext umfasst die Chipkartendienste. Diese steuern die direkte Kommunikation mit der Chipkarte. Sie basieren auf dem Open Card Framework [4]. Das Open Card Framework stellt eine grundlegende Treiberstruktur zum Zugriff auf Chipkarten über Java bereit. Auch eine Erweiterung in CardServices wird beschrieben, welche mit einem Cardlet auf der Chipkarte kommunizieren.

Im Campuskartenprojekt ist dies der `CampuskarteAPDUAppletProxy` (siehe Abbildung 5.1). Zum Management der Chipkarte muss mit einem anderen Cardlet auf der Chipkarte, dem Card-Manager, kommuniziert werden. Dies geschieht über die Klassen `VOPAuthCardProxy` zur Authentisierung, `VOPAppletManagerProxy` zum Aufspielen und Löschen von Cardlets, sowie dem `VOPAppletAccessProxy` für weitere Funktionalitäten, wie z.B. den Karteninhalt aufzulisten. Diese drei Dienste wurden als generische Schnittstellen im Open Card Framework bereits vorgegeben. Weiterhin gibt es eine Schnittstelle `Credential` zur Aufnahme von Authentisierungsinfor-

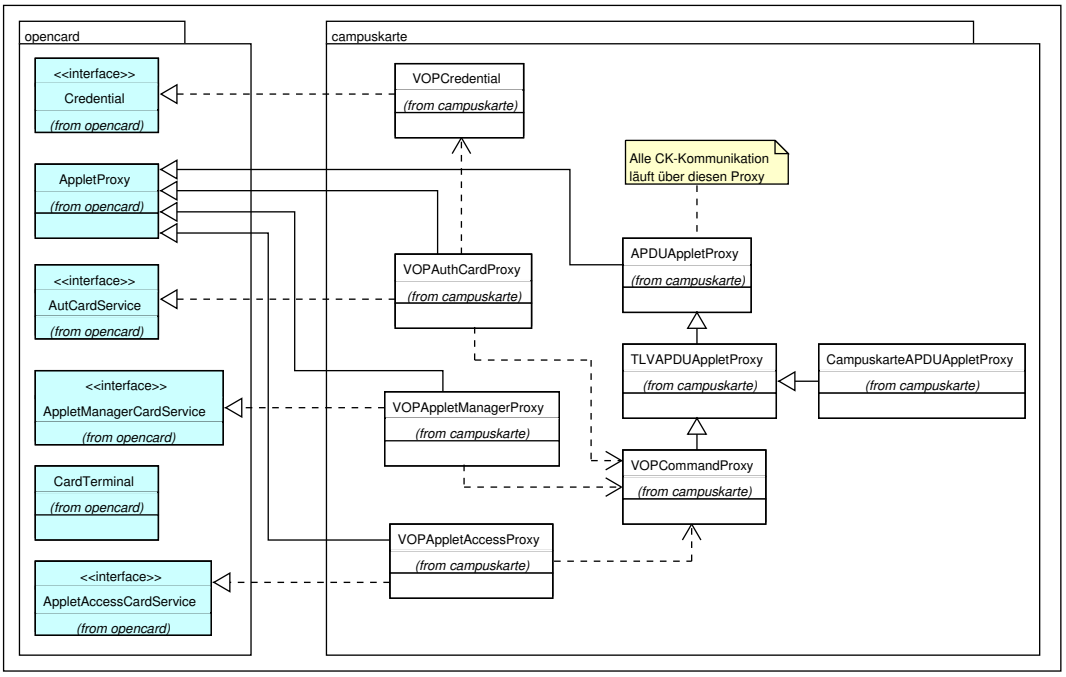

Abbildung 5.1: Relevante vorhandene Klassen des Campuskarten-Frameworks

mationen. Alle CardServices müssen der Klasse `AppletProxy` entstammen, welche einen Kommunikationskanal zu einem Cardlet bereitstellt. Der Name AppletProxy ist in diesem Zusammenhang etwas verwirrend, korrekt müsste er CardletProxy heißen. Die Klasse `CardTerminal` implementiert einen physikalischen Treiber für einen Chipkartenleser. Dieser ist vom Anwendungskontext nur für Spezialanwendungen, welche chipkartenleserabhängig sind (wie z.B. sichere PIN-Eingabe) notwendig. Die Klasse `VOPCommandProxy` im Campuskarten-Framework implementiert Kommandos für den Card-Manager nach der Visa Global Platform Spezifikation.

Das beschriebene Verfahren ist eine sehr starke Vereinfachung der wirklichen Vorgehensweise. Prinzipiell müssen ein oder mehrere der oben genannten Klassen für jeden möglichen Kartentyp überladen werden. Eine Kartentyperkennung initialisiert die entsprechende Unterklasse für die jeweilige Karte. Damit eine korrekte Kommunikation möglich wird, erfolgt diese nicht über die direkte Oberklasse `AppletProxy` der verschiedenen Services, sondern über den `VOPCommandProxy`, welcher wiederum als CardService von der Klasse `AppletProxy` abstammt. Somit ist gewährleistet, das alle implementierten CardServices die Funktionalität des VOPCommandProxy nutzen können. Sämtliche Kommunikation mit einem Cardlet läuft über die Klasse APDUAppletProxy. Eine genaue Beschreibung des Verfahrens ist in [36] enthalten.

Wie dem Systemüberblick im Kapitel 4.1.3 entnommen werden kann, soll das zu entwickelnde System auf einer Client-Server Architektur beruhen. Der Client soll nur die Anbindung an den Chipkartenleser sowie eine Ausgabemöglichkeit zur Verfügung stellen. Sämtliche Kommandos für die Chipkarte sollen auf dem Server generiert und ausgewertet werden. Um größtmögliche Kompatibilität und Sicherheit zu gewähren, soll sämtliche Kommunikation über das HTTPS Protokoll getunnelt werden. Das bedeutet aber auch, dass der Client eine Anfrage an den Server schicken muss, es kann keine eigene Anfrage des Servers an den Client durchgeführt werden. Zu diesem Zweck ist eine Trennung zwischen Befehlsgenerierung und Auswertung für die Chipkarte unerlässlich. Der Client fordert im ersten Schritt einen neuen Befehl für die

Chipkarte an und leitet das Ergebnis der Chipkarte im zweiten Schritt an den Server weiter. Um dies zu realisieren müsste man den `APDUAppletProxy` „aufschneiden“ und diese Art der Kommunikation einbauen. Ein wesentliches Problem würde sich aber in der Initialisierung der Komponenten des Open Card Frameworks ergeben. Das Basis-Framework müsste auf dem Client, ein davon abhängiger Teil jedoch auf dem Server gestartet werden.

Das bereits vorliegende Framework hat viele weitere Nachteile, wie z.B. eine nicht vorhandene Schlüssel- oder Applikationsverwaltung. Der Benutzer muss den Master-Schlüssel für alle Chipkarten besitzen, um auf diese zugreifen zu können. Verschiedene Schlüssel für identische Kartentypen sind nicht vorgesehen. Es können weder Meta-Informationen über auf der Chipkarte vorhandene Cardlets, noch über neu installierbare Cardlets für einen bestimmten Kartentyp abgerufen werden. Auch ein möglicher Kartenstatus wird nicht unterstützt, wäre aber in Anbetracht der Kartenmengen sehr wichtig (siehe Anforderungsanalyse).

Zu den genannten Nachteilen des vorliegenden Frameworks kommt eine Implementierung, welche ich „evolutionär gewachsen“ nennen würde. Der Quelltext ist durch unnötigen Ballast für ehemals testweise unterstützte, aber nicht mehr benötigte Chipkarten enorm aufgebläht. Die Integration neuer Chipkarten ist sehr aufwendig, auch wenn sich diese nur in kleinen Details von bereits vorhandenen Karten unterscheiden. Ebenso existieren Inkompatibilitäten mit verschiedenen Chipkartenlesern, die darauf zurückzuführen sind, dass das vorhandene Framework Kommunikationsschichten nachbildet, die bereits im OCF enthalten sind.

Aufgrund der genannten Nachteile und durch meine Kenntnisse über das vorhandene System, bin ich zu der Feststellung gekommen, dass erst ein komplettes Redesign die neuen Anforderungen erfüllen und eine stabile Basis für die zukünfige Entwicklung bilden kann. Ich habe diese Entscheidung nicht leichtfertig getroffen, da das Redesign einen großen Teil der prototypischen Implementierung ausmacht. Dennoch kann ein in den Anforderungen genanntes Framework nicht ohne eine solide Basis existieren, für die ein komplettes Redesign der Chipkartendienste eine gute Voraussetzung ist.

5.3 Anwendungsfälle

Einen detaillierten Überblick über alle für diese Arbeit relevanten Anwendungsfälle gibt Abbildung 5.2. Diese werden in den folgenden Abschnitten nach Aktoren gruppiert und genauer spezifiziert. Nicht alle Anwendungsfälle sind implementationstechnisch möglich (z.B. „Karte ausgeben“) oder im Rahmen dieser Arbeit implementiert (z.B. „Zertifikate bereitstellen“). Da sie jedoch als Teile eines übergeordneten Modells gesehen werden können sind sie zum Verständnis unerlässlich. Alle Anwendungsfälle wurden durch eine farblich Kennzeichnung bestimmten Aktoren zugeordnet, diese wird auch in den Aktivitätsdiagrammen verwendet.

5.3.1 Karten–Halter

Für den Karten–Halter als Benutzer der Chipkarte sind folgende Anwendungsfälle relevant:

- **Karteninhalt anzeigen**
  Listet den Inhalt der Chipkarte des Karten-Halters auf. Die installierten Cardlets werden über den Anwendungsfall VOP Dienste bereitstellen ermittelt und

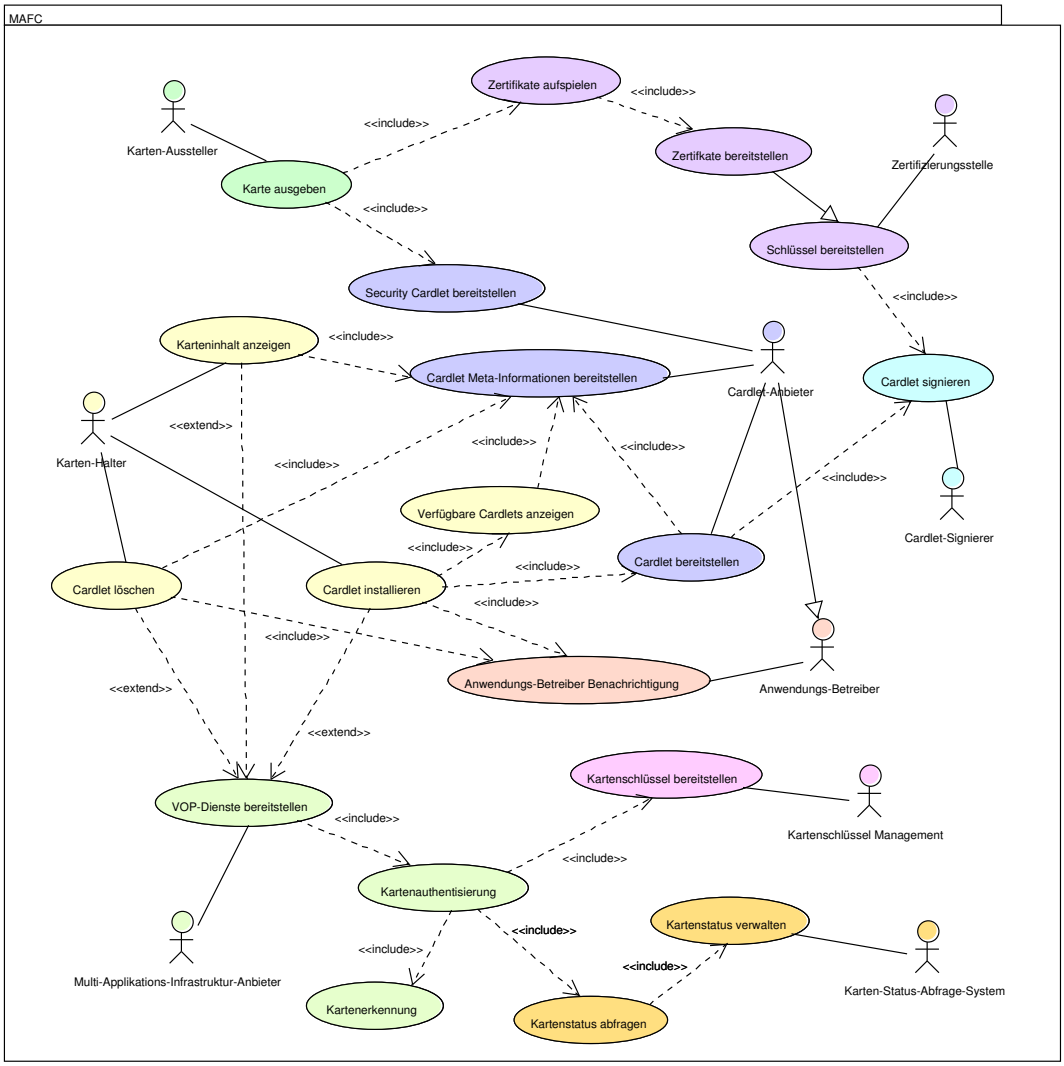

Abbildung 5.2: Übersicht über alle Anwendungsfälle

anschließend mit den Informationen von Cardlet Meta-Informationen bereitstellen aufbereitet.

- **Verfügbare Cardlets anzeigen**
  Zeigt die zur Installation verfügbaren Cardlets an. Diese Information wird aus dem Anwendungsfall Meta-Informationen bereitstellen ermittelt.

- **Cardlet installieren**
  Installiert ein Cardlet auf der Chipkarte des Karten–Halters. Dies umfasst verschiedene Aktivitäten, welche in Abbildung 5.3 dargestellt sind.

- **Cardlet löschen**
  Löscht ein Cardlet auf der Chipkarte des Benutzers. Dies umfasst verschiedene Aktivitäten, welche in Abbildung 5.4 dargestellt sind.

5.3.2 Multi–Applikations–Infrastruktur–Anbieter

Der Multi–Applikations–Infrastruktur–Anbieter stellt den eigentlichen Dienst zum Rekonfigurieren der Chipkarte zur Verfügung.

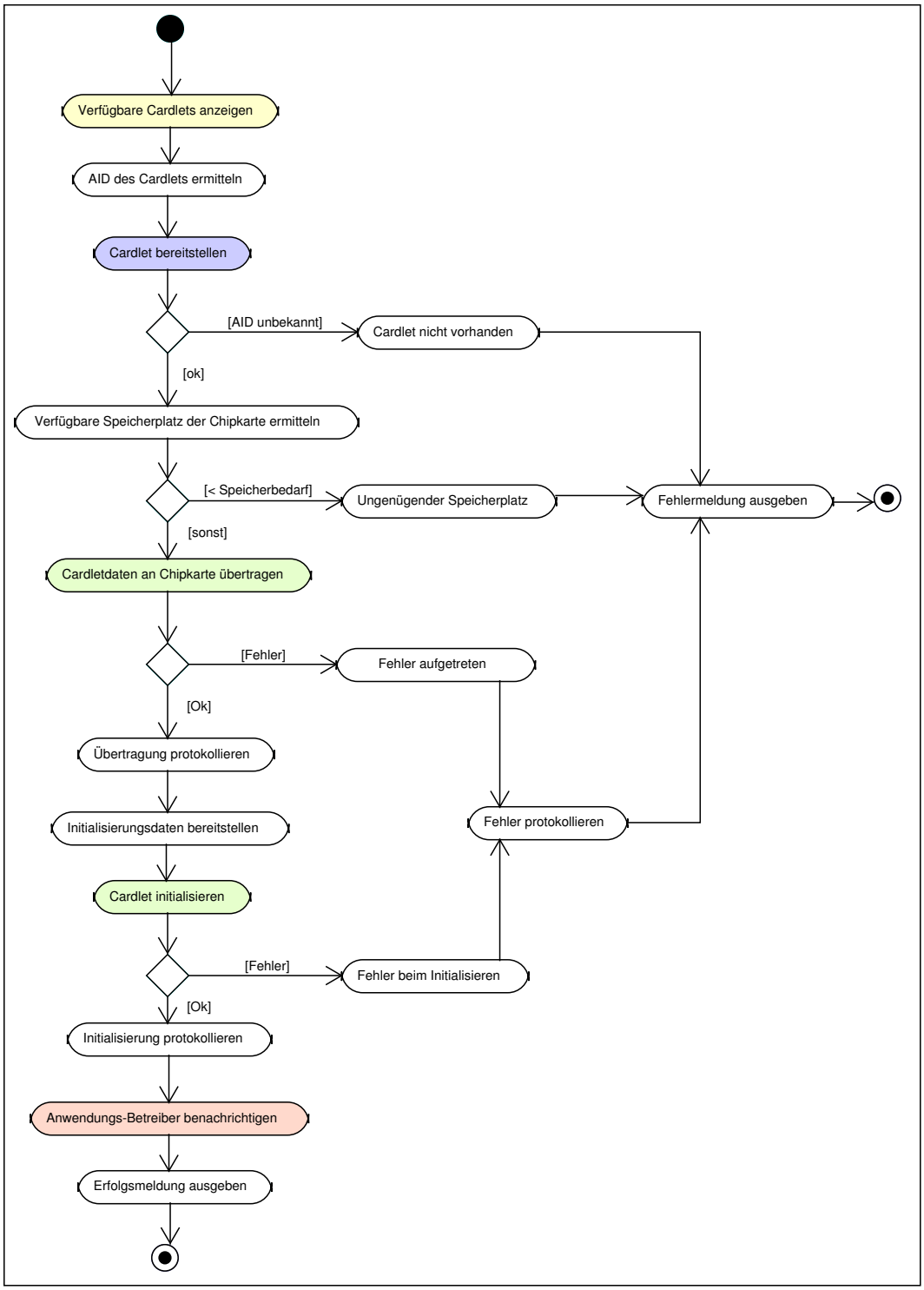

Abbildung 5.3: Aktivitätsdiagramm: Cardlet installieren

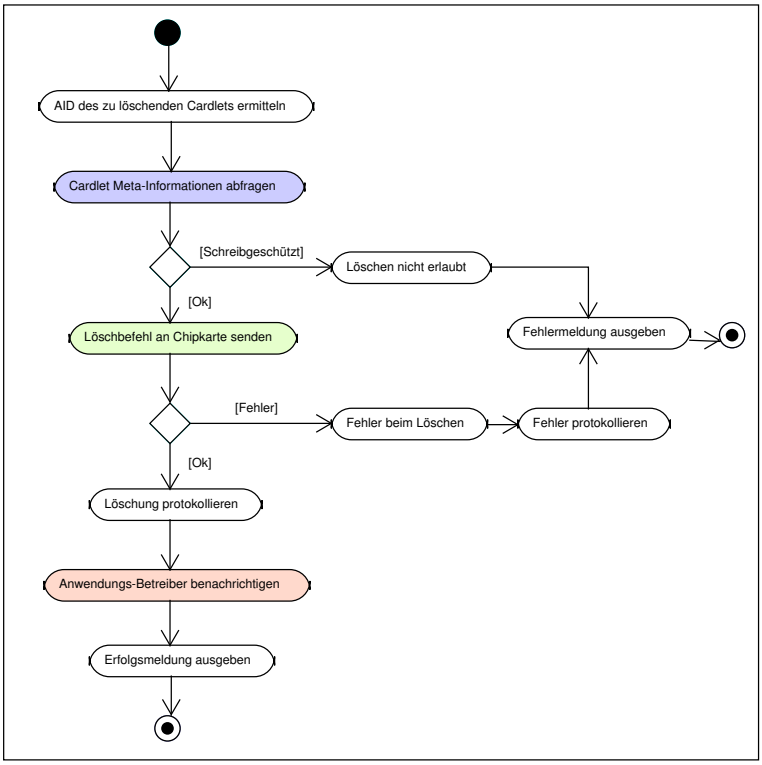

Abbildung 5.4: Aktivitätsdiagramm: Cardlet löschen

- **VOP–Dienste bereitstellen**
  Stellt VOP 2.0.1 Kommandos zum Kartenmanagement zur Verfügung. Eine Kartentyp-spezifische Anpassung dieser Kommandos erfolgt über den Anwendungsfall Kartenerkennung. Bevor die Kommandos verwendet werden können, muss eine Authentisierung des Card-Managers erfolgen. Dies erfolgt über den Anwendungsfall Kartenauthentisierung.

- **Kartenerkennung**
  Erkennt den Typ der eingelegten Karte und lädt kartenspezifische Klassen. Siehe Abbildung 5.5.

- **Kartenauthentisierung**
  Authentisiert den Card–Manager der Chipkarte. Dies umfasst verschiedene Aktivitäten, welche in Abbildung 5.6 dargestellt sind.

5.3.3 Cardlet–Anbieter

Der Cardlet–Anbieter stellt Cardlets und Meta-Informationen zur Verfügung.

- **Security Cardlet bereitstellen**
  Ein Security Cardlet ist eine Security–Domain auf einer Chipkarte, die Signaturen von Cardlets prüfen sowie signierte Bestätigungen erstellen kann. Dieser Anwendungsfall ist für zukünftige Erweiterungen gedacht.

- **Cardlet Meta–Informationen bereitstellen**
  Stellt Meta–Informationen über Cardlets bereit. Zu einer AID können so z.B.

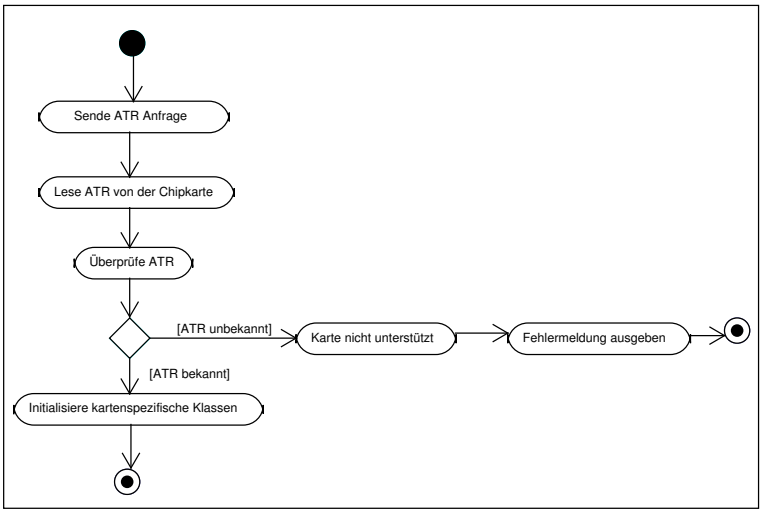

Abbildung 5.5: Aktivitätsdiagramm: Kartenerkennung

Name, Größe, Herausgeber sowie die digitale Signatur ermittelt werden. Weiterhin muss eine Liste aller bekannten AID's bereitgestellt werden.

- **Cardlet bereitstellen**
  Stellt ein Cardlet zum Installieren auf der Chipkarte zur Verfügung. Dies umfasst die Anwendungsfälle Cardlet Meta–Informationen bereitstellen sowie Cardlet signieren.

5.3.4 Anwendungs–Betreiber

Im Rahmen des Multi–Applikations–Frameworks für Chipkarten (MAFC) ist folgender Anwendungsfall für den Anwendungs–Betreiber relevant:

- **Anwendungs–Betreiber benachrichtigen**
  Benachrichtigt den Anwendungs–Betreiber über das Installieren oder Löschen eines Cardlets. Dieser Anwendungsfall kann zur (De)–Aktivierung von Diensten sowie zu Abrechnungszwecken verwendet werden.

5.3.5 Karten–Aussteller

Der Karten-Aussteller dient im Rahmen dieser Arbeit nur dem Verständnis, seine Anwendungsfälle werden nicht implementiert.

- **Karte ausgeben**
  Bereitet eine Karte zur Ausgabe an den Karten-Halter vor. Dies umfasst die Anwendungsfälle Zertifikate aufspielen sowie Security Cardlet bereitstellen.

- **Zertifikate aufspielen**
  Lädt das Zertifikat des Cardlet-Signierers in das Security Cardlet. Dies muss auf einem sicheren Weg vor der Herausgabe der Chipkarte erfolgen. Mit Hilfe des Zertifikates kann die digitale Signatur von Cardlets überprüft werden.

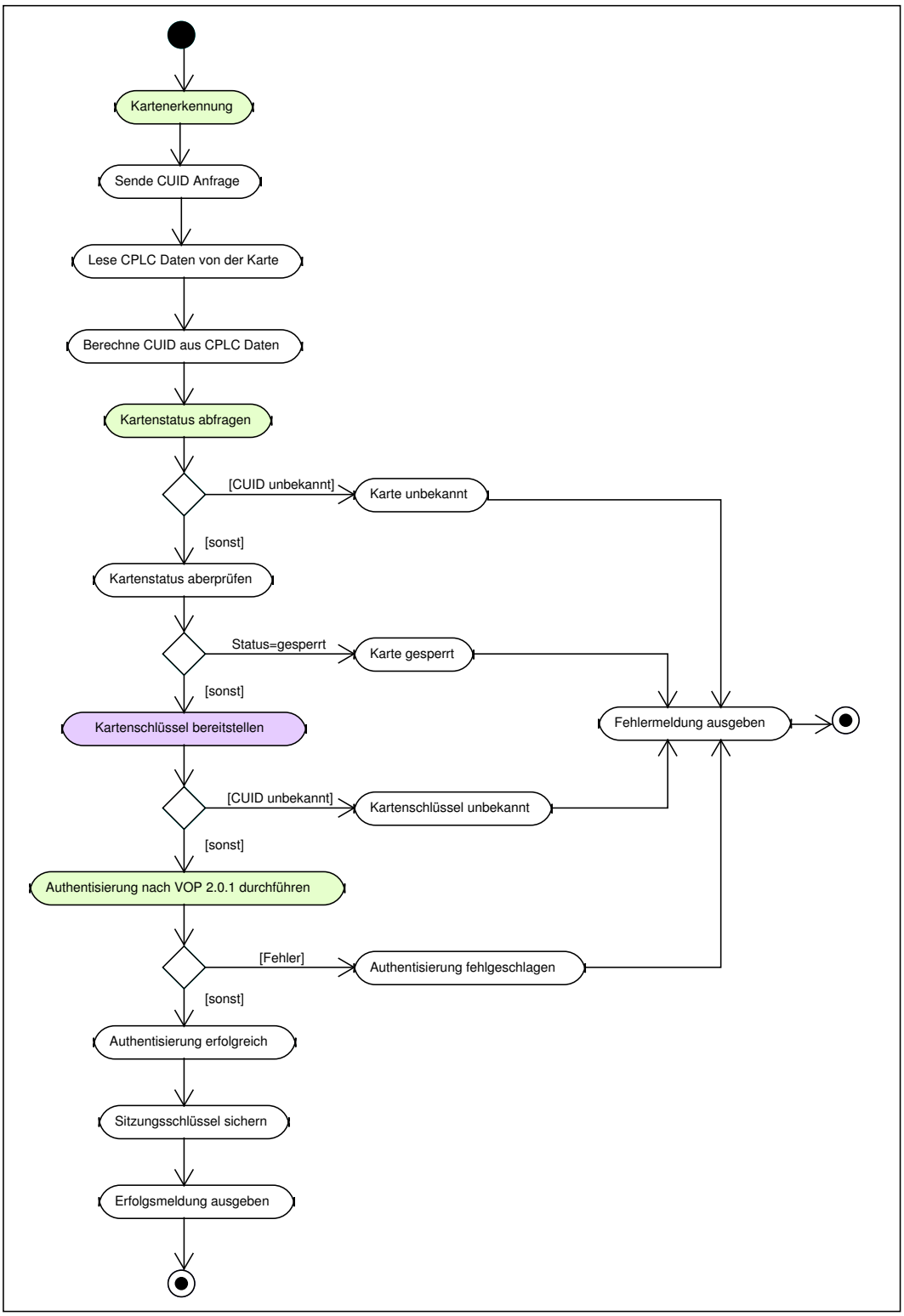

Abbildung 5.6: Aktivitätsdiagramm: Kartenauthentisierung

5.3.6 Zertifizierungsstelle

Die Zertifizierungsstelle dient im Rahmen dieser Arbeit nur dem Verständnis, ihre Anwendungsfälle werden nicht implementiert.

- **Schlüssel bereitstellen**
  Dieser Anwendungsfall erzeugt einen privaten Schlüssel für den Cardlet–Signierer.
- **Zertifikate bereitstellen**
  Dieser Anwendungsfall erzeugt ein öffentliches Zertifikat vom Cardlet-Signierer. Zertifikate bereitstellen ist eine Spezialisierung von Schlüssel bereitstellen.

5.3.7 Cardlet-Signierer

Der Cardlet-Signierer dient im Rahmen dieser Arbeit nur der zukünftigen Erweiterbarkeit, sein Anwendungsfall wird nicht implementiert.

- **Cardlet signieren**
  In diesem Anwendungsfall prüft der Cardlet-Signierer ein Cardlet nach spezifischen Kriterien und signiert es anschließend digital. Dazu wird der Anwendungsfall Schlüssel bereitstellen benötigt.

5.3.8 Kartenschlüssel–Management

Das Kartenschlüssel–Management dient der Bereitstellung verschiedener kryptographischer Schlüssel für Chipkarten.

- **Kartenschlüssel bereitstellen**
  In diesem Anwendungsfall werden die kryptographischen Schlüssel für den Card–Manager einer bestimmten Chipkarte abgefragt.

5.3.9 Karten–Status–Abfrage–System

Im Rahmen des MAFC sind folgende Anwendungsfälle für das Karten–Status–Abfrage–System relevant:

- **Kartenstatus verwalten**
  Verwaltet den Status einer Karte. Im Rahmen der prototypischen Implementierung werden nur vier Zustände verwendet: `ok`, `gesperrt`, `zurückgezogen` sowie `unbekannt`.
- **Kartenstatus abfragen**
  Fragt den Status einer Karte ab. Dies umfasst Kartenstatus verwalten.

5.4 Sicherheitsanalyse

Die Sicherheitsanalyse beschreibt die Schutzziele verschiedener, mit dem System interagierender, Aktoren sowie mögliche Angreifer mit ihren Interessen und typischen Fähigkeiten.

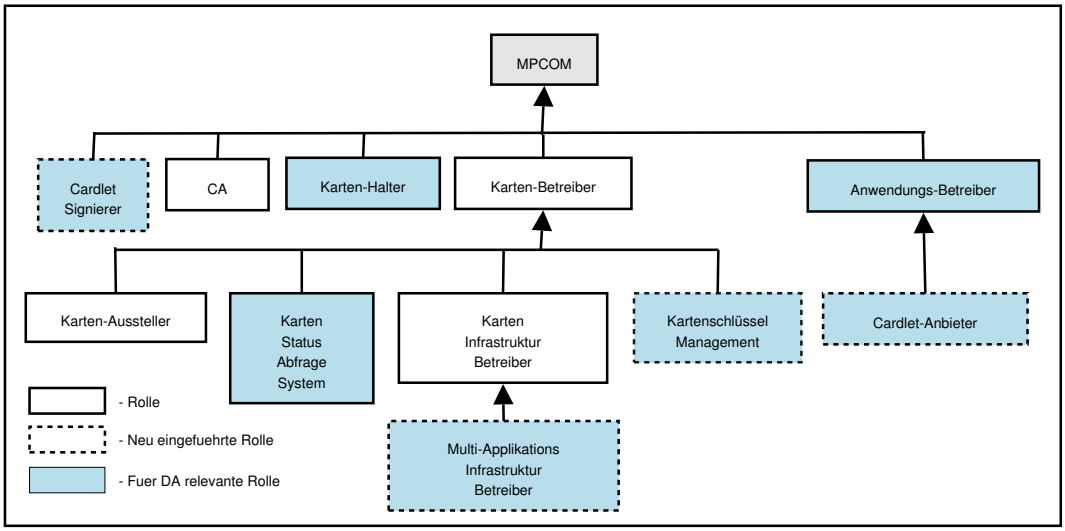

Abbildung 5.7: Erweiterte Rollendefinitionen im Multi Party Card Operating Model

5.4.1 Schutzziele

Um individuelle Schutzziele festlegen zu können, muss zuerst festgestellt werden, welche Personen in welchen Rollen mit dem System zu tun haben werden.

Identifizierte Rollen

Folgende Rollen wurden für das Multi-Applikations-Framework für Chipkarten anhand der Anwendungsfälle identifiziert. Eine andere Gliederung ist in Abbildung 5.7 dargestellt. Diese basiert auf einem Multi–Party–Card–Operating–Model (MPCOM) für Chipkarten, welches von Thomas J. Wilke und mir im Rahmen von EU–Projektanträgen skizziert wurde.

A : **Karten–Betreiber** – Unter dem Begriff Karten–Betreiber werden folgende Einzelrollen zusammengefasst:

1. **Multi–Applikations–Infrastruktur–Anbieter** – Der Multi–Applikations–Infrastruktur–Anbieter stellt die Kommandos zur Rekonfiguration der Chipkarte bereit.
2. **Karten–Status–Abfrage–System** – Das Karten–Status–Abfrage–System dient der Abfrage von Kartenstatus–Zuständen.
3. **Kartenschlüssel–Management** – Das Kartenschlüssel–Management verwaltet verfügbare Kartenschlüssel.

B : **Karten–Halter** – Der Karten–Halter möchte seine Karte komfortabel von überall rekonfigurieren können. Dies umfasst die Ansicht des Karteninhalts sowie das Löschen und Installieren von Cardlets. Der Karten–Halter möchte die Kontrolle darüber behalten, was auf seine Karte geladen wird.

C : **Cardlet–Signierer** – Der Cardlet–Signierer prüft den Quellcode eines Cardlets und unterschreibt den geprüften Code anschließend digital.

D : **Anwendungs–Betreiber** – Der Anwendungs–Betreiber kommuniziert mit Cardlets auf der Karte, um deren Dienste zu nutzen.

E : **Cardlet–Anbieter** – Der Cardlet–Anbieter stellt digital signierte Cardlets zum Aufspielen auf die Karte bereit. Da diese Cardlets der On–Card Teil einer Anwendung sind, ist der Cardlet–Anbieter eine Teilmenge eines zugehörigen Anwendungs–Betreibers.

F : **Unbeteiligte** – Dies sind alle, die nicht in eine der obigen Rollen zugeordnet werden können. Darunter fallen Angreifer ebenso wie unbeteiligte Dritte.

Erkannte Schutzziele

Folgende Schutzziele sind im Rahmen der Arbeit als sinnvoll erkannt worden. Es treten keine sich widersprechenden Schutzziele auf.

- **Integrität** – Das höchste Schutzziel im Rahmen dieser Arbeit ist die Integrität. Dies betrifft alle an der jeweiligen Kommunikation beteiligten Rollen. Eine besondere Beachtung muss jedoch der Integrität der Verbindung zwischen dem Multi–Applikations–Infrastruktur–Anbieter und dem Karten–Halter gewährt werden.
- **Vertraulichkeit** – Der Karten–Betreiber und der Karten–Halter möchten die Vertraulichkeit ihrer Kommunikation durchgesetzt wissen.
- **Verbindlichkeit** – Der Cardlet–Signierer möchte die Verbindlichkeit seiner Aussagen gegenüber anderen Rollen nachweisen können.
- **Weitere Schutzziele** – Weitere Schutzziele, wie z.B. Anonymität oder Verfügbarkeit liegen außerhalb des Rahmens dieser Arbeit.

5.4.2 Angreifermodell

Dieser Abschnitt listet einige mögliche Angreifer auf. Es wurde eine Unterteilung in die einzelnen Schutzziele vorgenommen. Mögliche Angreifer können natürlich auch mehrere Schutzziele gleichzeitig durchbrechen.

Angriffe gegen das Schutzziel Integrität

Angreifer, welche das Schutzziel Integrität durchbrechen wollen, haben üblicherweise Interesse an der Manipulation übertragener Daten. Sie können sich als eine Rolle ausgeben, die sie gar nicht innehaben, um den Kommunikationspartnern falsche Daten vorzutäuschen oder fremden Programmcode einzuschleusen. Dadurch können sowohl der Karten–Halter als auch der Karten–Betreiber schwer geschädigt werden. Bei einem Angriff kann der Karten–Halter nicht überprüfen, ob seine Kommunikation mit dem Karten–Betreiber korrekt verläuft. Zudem kann der Karten–Betreiber einen inkonsistenten Zustand erlangen, da er gefälschte Angaben zu Aktivitäten des Karten–Halters erhalten kann. Ein direkter Angriff auf die Kommunikation Karten–Betreiber vs. Cardlet–Anbieter könnte bereits in diesem Schritt zur Einschleusung von Fremdcode führen. Ein Angreifer innerhalb der Rolle Karten–Betreiber könnte den Einsatz des Systems vollständig kompromittieren.

Angriffe gegen das Schutzziel Vertraulichkeit

Angreifer könnten ein besonderes Interesse an der Kommunikation zwischen Karten–Halter und Karten–Betreiber besitzen. Damit ist es ihnen möglich, den kompletten Rekonfigurationsvorgang zu belauschen. Sie erfahren welche Anwendungen der Karten–Halter benutzt und wie er diese konfiguriert hat.

Ein spezieller Angriff könnte innerhalb der Rolle Karten–Betreiber erfolgen. Bei einer möglichen Belauschung der Kommunikation zwischen Kartenschlüssel–Management und Multi–Applikations–Infrastruktur–Anbieter kann ein Angreifer geheime Kartenschlüssel in Erfahrung bringen. Mit diesen Schlüsseln wäre er in der Lage, die Chipkarte des Karten–Halters zu manipulieren.

Angriffe gegen das Schutzziel Verbindlichkeit

Ein möglicher Angreifer könnte die digitale Signatur von Cardlets durch den Cardlet–Signierer fälschen, wodurch Fremdcode in das System eingeschleust werden könnte. Wird dieser Angriff nicht erkannt, so kann es nicht nur zur schwerwiegenden Schädigung des Systems führen. Ebenfalls möglich wären auch rechtliche Schritte gegen den Cardlet–Signierer, da dieser für die Prüfung der Cardlets verantwortlich und haftbar ist.

Angriffe gegen weitere Schutzziele

Angreifer können weitere Schutzziele mittels anderer oder neuer Angriffsarten durchbrechen [39]. Ein Schutz vor diesen Angriffen liegt außerhalb dieser Arbeit.

Kapitel 6

Systementwurf

Im Systementwurf werden konkrete Lösungen für die in der Systemanalyse erarbeiteten Anwendungsfälle beschrieben. Im ersten Schritt wird der grundlegende Systemaufbau detailliert erläutert und eine Trennung zwischen den verschiedenen Komponenten vorgenommen. Diese werden in den folgenden Unterabschnitten mit Hilfe von Klassendiagrammen[1] sowie, wo es sich anbietet, Zustandsdiagrammen beschrieben. Die Kommunikation zwischen den beiden Hauptkomponenten wird in einem eigenen Unterkapitel betrachtet. Entsprechende Sicherheitsbetrachtungen werden am Ende eines jeden Unterkapitels vorgenommen.

Zum Anfang der Entwurfsbetrachtungen ist jedoch die Frage zu klären: Was ist ein Framework? Oder besser gesagt: Was ist ein objektorientiertes Framework?[2]

Frameworks basieren auf dem Gedanken der modulbasierten Programmierung. Dabei werden verschiedene, immer wieder benötigte Routinen in einzelne Programmmodule gepackt. Sollen diese später weiterverwendet werden, so ist entweder eine Anpassung des Quelltextes oder die Erzeugung eines Wrappers notwendig. Die objektorientierte Programmierung führte viele Neuerungen im Bereich Wiederverwendung ein, so können Wrapper z.B. sehr einfach durch Spezialisierung erzeugt werden.

Leider gibt es bis heute keine eindeutig anerkannte Definition eines objektorientierten Frameworks. Eine zu dieser Arbeit sehr gut passende findet sich in [44]:

„A framework is a partial design and implementation for an application in a given problem domain.“

Der Teil dieser Arbeit, welcher sich sehr gut als Framework entwickeln lässt, deckt den Bereich der Dienste ab. Andere Teile, wie z.B. der Server, stellen die ersten Anwendungen des Frameworks dar. Die Dienste können von verschiedenen Anwendungen genutzt werden, müssen jedoch auch sehr flexibel und erweiterbar bleiben. So soll z.B. die Anpassung an neue Chipkarten möglichst ohne die Änderung bereits existierenden Quelltextes möglich sein.

Zur Erfüllung der genannten Anforderungen habe ich ein Programmmuster (Pattnern) entworfen. Dieses ermöglicht die flexible Bereitstellung verschiedener Dienste innerhalb einer Sitzung. Eine nähere Erläuterung folgt im Abschnitt Dienste. Andere bekannte Programmmuster wurden verwendet, so z.B. Iterator, Mediator oder State [51]. Diese wurden teilweise in flexibel konfigurierbare Formen abgewandelt, so können z.B. Observer über Konfigurationsdateien angemeldet werden, anstatt diese

[1] Bei den Klassen– und Schnittstellendiagrammen wurde aus Platzgründen auf die Darstellung der Parameter verzichtet. Diese können der Java-Klassendokumentation entnommen werden.

[2] Die Implementierung erfolgt in der objektorientierten Sprache Java.

Dienst/Anwendung	Package
Multi-Applikations-Infrastruktur-Anbieter	`campuskarte.mafc.maip`, `campuskarte.mafc.maip.structures`, `campuskarte.mafc.maip.manufacturer`
Karten-Status-Abfrage-System	`campuskarte.mafc.csa`
Cardlet-Anbieter	`campuskarte.mafc.cp`, `campuskarte.mafc.cp.structures`
Anwendungs-Betreiber	`campuskarte.mafc.sp`
Kartenschlüssel-Management	`campuskarte.mafc.kp`
Server	`campuskarte.mafc.server`, `campuskarte.mafc.server.servlets`, `campuskarte.mafc.server.states`
Client	`campuskarte.mafc.client`, `campuskarte.mafc.client.frames`

Tabelle 6.1: Übersicht über die Packages des MAFC

direkt im Quelltext zu manifestieren.

Komponenten

Anhand des grundlegenden Systemaufbaus aus Kapitel 4.1.3 lässt sich bereits eine Unterteilung des Systems in verschiedene Komponenten ermitteln. Die wichtigste Komponente sind verschiedene Dienste, welche jeweils spezielle Aufgaben erfüllen. Zur Nutzung dieser Dienste ist die Programmierung spezieller Geräte notwendig. Dies sind im einzelnen ein Server, die dazugehörigen Clients sowie die Chipkarten. Ein Server kann dabei üblicherweise mehrere Clients gleichzeitig bedienen, pro Client darf es aber nur eine aktive Chipkarte geben.

In Tabelle 6.1 werden die verwendeten Packages der Komponenten des Multi–Applikation–Frameworks für Chipkarten angegeben. Die Tabelle enthält nur Komponenten, welche implementiert wurden.

6.1 Grundlegende Kommunikation

In Abbildung 6.1 ist die grundlegende Kommunikation zwischen den Komponenten dargestellt. Die Grafik fasst drei Abstraktionsebenen zusammen. Die oberste und höchste Ebene ist die der Dienste, welche unabhängig von Transportprotokollen und physischen Gegebenheiten existieren können. Darunter folgt die Schicht der physikalisch vorhandenen Geräte, von welchen eines, der Web–Server, die Dienste benutzt. Wie er dies tut, ist nicht dargestellt. Im unteren Teil der Abbildung folgen verschiedene Protokolle und andere Schichten, welche direkt der Kommunikation dienen. Die Abbildung 6.1 wird im Folgenden näher erläutert.

Der Client schickt eine Anfrage an den Web-Server, welcher daraufhin eine APDU–Sequenz von den Chipkartendiensten erhält. Diese APDU-Sequenz wird über verschiedene Schichten an den Client übertragen und an die Chipkarte gesendet. Die Antwort-APDU der Chipkarte wird zurück an den Web-Server gesendet, der diese den Chipkartendiensten zur Auswertung übergibt. Die Chipkarten- sowie alle anderen Dienste sind in der prototypischen Implementierung fest mit dem Web-Server

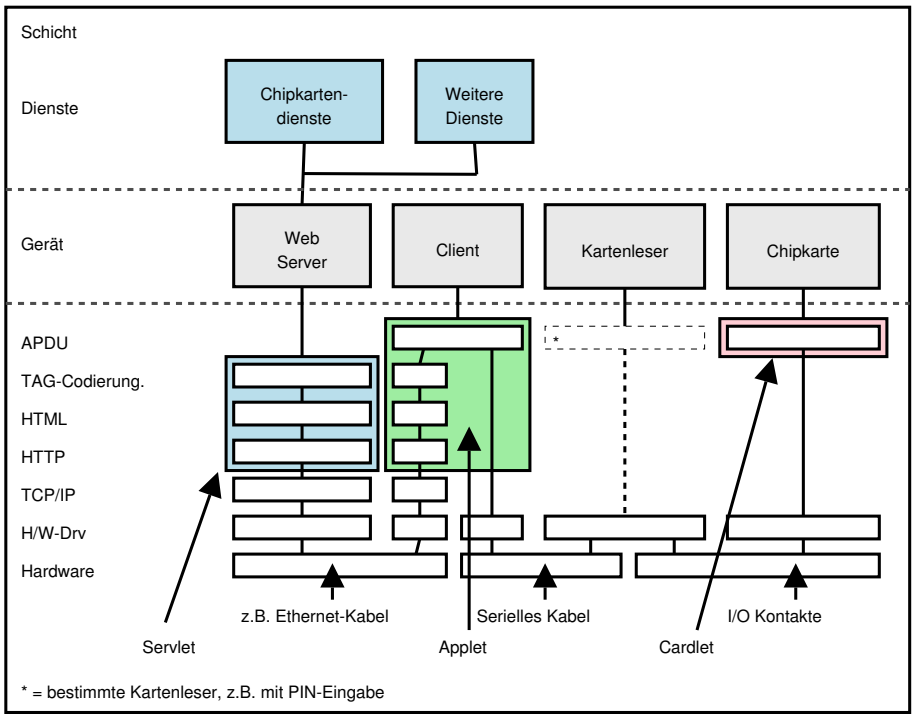

Abbildung 6.1: Kommunikation zwischen den Komponenten

verbunden. Durch ein flexibles Schnittstellenkonzept ist dies jedoch leicht erweiterbar (siehe Abschnitt 6.2.1). Zusätzlich zu den Chipkartendiensten benötigt der Web-Server auch weitere Dienste, wie das Karten–Status–Abfrage–System. Diese sind in der Abbildung als weitere Dienste dargestellt, da sie zur Kommunikation nicht notwendig sind. Alle Dienste werden in den folgenden Unterabschnitten beschrieben. Eine genauere Betrachtung der Kommunikation zwischen Server und Client wird in einem eigenen Abschnitt betrachtet.

6.2 Dienste

Die verschiedenen Dienste stellen den Kernpunkt dieser Arbeit dar. Sie bilden die Grundlage für ein erweiterbares und flexibles Framework für Multi–Applikations–Chipkarten. Eine Teilmenge der Dienste, die Chipkartendienste, lagen bereits in einer vorhandenen Implementierung vor. Warum diese nicht übernommen werden konnten, wurde bereits in Kapitel 5.2 erläutert. Im Folgenden werden die Anwendungsfälle aus Abbildung 5.2 einem Dienst zugeordnet. Dies bezieht sich auf alle Anwendungsfälle, welche implementiert werden müssen. Die Anwendungsfälle des Karten-Halters werden nicht als Dienste, sondern vom Server unter der Nutzung der Dienste implementiert. Zuvor jedoch wird das Konzept zur Verwendung der Dienste erläutert.

6.2.1 Service–Handler Konzept

Um das Ziel eines flexiblen und erweiterbaren Frameworks zu erreichen, sollen alle Funktionen der Dienste über Schnittstellen definiert werden. Bei Erweiterung des Frameworks auf neue Funktionalität müssen, um abwärtskompatibel zu bleiben,

neue Schnittstellen eingeführt werden. Eine Implementation kann sowohl alte, als auch neue Schnittstellen in einer Klasse vereinen. Die konkreten Implementierungen einer Schnittstelle sollen über einen so genannten Service–Handler (Dienst-Anbieter) erhältlich sein. Jede Instanz eines Service–Handlers stellt eine eigene (neue) Sitzung zur Verfügung. Die Implementierung der Schnittstellen soll bei der Instanzierung eines neuen Service–Handler Objektes frei konfigurierbar gehalten werden.

Ein Beispiel zur Verdeutlichung des Konzeptes: Eine neue Java-Anwendung möchte gern Dienste des Multi–Applikations–Frameworks für Chipkarten nutzen. Sie muss dazu Zugriff auf die Service–Handler Klasse haben. Von dieser erstellt sie eine neue Instanz. Mit Hilfe dieser Instanz kann sie sich jetzt alle implementierten Schnittstellen des Frameworks zurückgeben lassen. Wie und wo diese implementiert sind, liegt außerhalb ihrer Zuständigkeit. Wie in der prototypischen Implementierung können dies weitere Java-Klassen sein, aber auch ein entfernter Server könnte angesprochen werden. Eine Umstellung auf neue oder alternative Implementierungen der Dienste hat somit nur eine Rekonfiguration des Service–Handlers zur Folge. Diese Rekonfiguration soll ohne Neuerstellung des Quelltextes möglich sein. Eine genaue Beschreibung der Implementation erfolgt in Kapitel 7.4.

6.2.2 Multi-Applikations-Infrastruktur-Anbieter

Der Multi-Applikations-Infrastruktur-Anbieter[3], definiert Schnittstellen für die folgenden Anwendungsfälle. Die Schnittstellen sind in Abbildung 6.2 dargestellt.

- **VOP-Dienste bereitstellen** - Wird definiert durch die Schnittstellen `IMAFCCommandProxy` zum Erzeugen von Kommandos (APDU's) für Chipkarten, `IMAFCEncryptionProxy` zur Verschlüsselung der Kommandos sowie `IMAFCAppletManagerProxy` zum Ausführen und Auswerten von höheren Operationen.
- **Kartenauthentisierung** - Wird definiert durch die Schnittstelle `IMAFCAutCardProxy` zur Kartenauthentisierung.
- **Kartenerkennung** - Wird definiert durch die Schnittstelle `IMAFCCardDetector` zur Erkennung des Chipkartentyps.

Der Schnittstelle `IMAFCCardDetector` kommt gleichzeitig die Aufgabe zu, die anderen hier aufgeführten Dienste des Multi–Applikations–Infrastrastruktur–Anbieters, abhängig vom Kartentyp, einzubinden. Das bedeutet, zum Anfang kann nur auf die Schnittstelle `IMAFCCardDetector` des Multi-Applikations-Infrastruktur-Anbieters zurückgegriffen werden. Erst nachdem diese mit einem bestimmten Kartentyp initialisiert wurde, sind die anderen Schnittstellen über den Service–Handler verfügbar. So ist gewährleistet, dass für jeden Chipkartentyp eine eigene Kommandogenerierungs-, Verschlüsselungs- sowie Authentisierungsfunktionalität zur Verfügung gestellt werden kann.

Die Schnittstelle `IMAFCCommandProxy` definiert die im Rahmen dieser Arbeit benötigte Funktionalität zum Zugriff auf eine VOP 2.0.1 kompatible Chipkarte. Dabei wurde aus Gründen des Client–Server Konzeptes eine Trennung zwischen Befehlsgenerierung und Befehlsauswertung vorgenommen.

Die Schnittstelle `IMAFCEncryptionProxy` definiert Verschlüsselungs- sowie Authentikationscodeerzeugung für die Kommunikation mit einer Chipkarte.

[3] in der Implementierung Multi Application Infrastructure Provider, kurz MAIP genannt

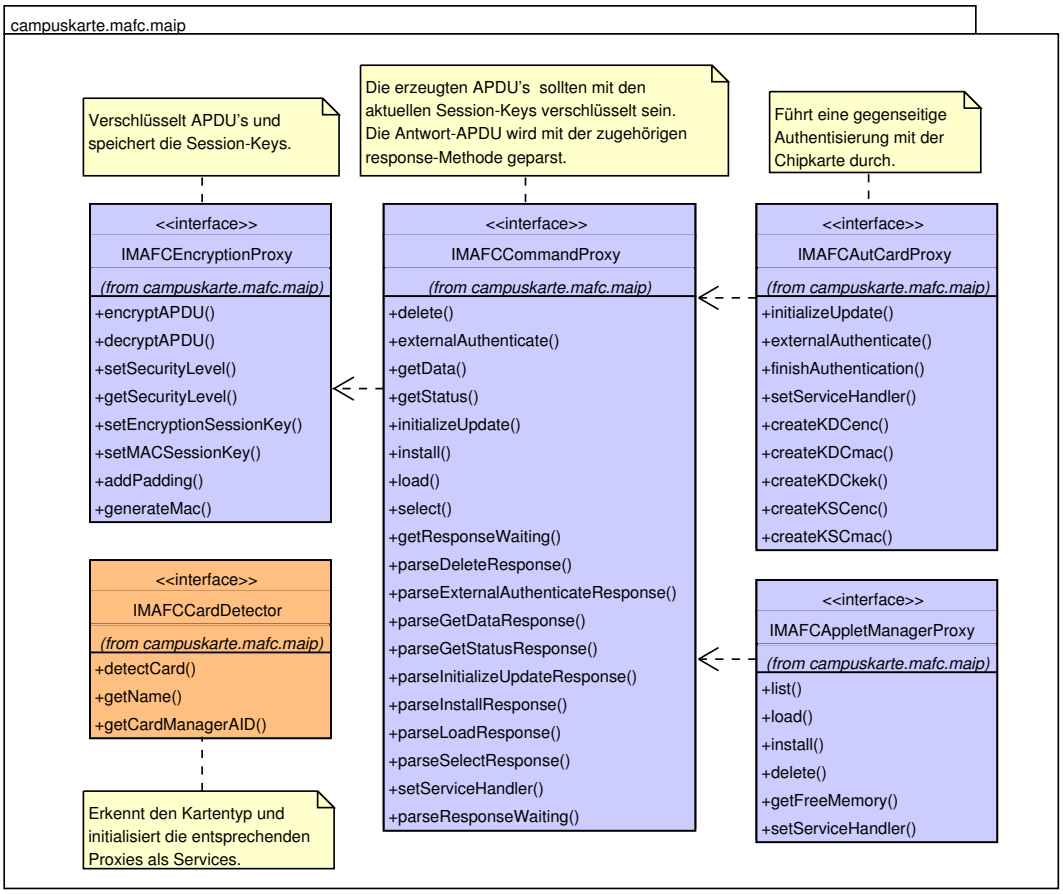

Abbildung 6.2: Schnittstellen für den Multi–Applikations–Infrastruktur–Anbieter

Die Schnittstelle `IMAFCAutCardProxy` definiert die Ausführung eines Authentisierungsprozesses für Visa Global Platform [19] kompatible Chipkarten. Dieser Prozess wird noch nicht von allen Herstellern einheitlich durchgeführt, so dass auch hier unter Umständen eine kartenspezifische Implementierung notwendig ist. Des Weiteren werden Methoden zur Ableitung von statischen Kartenschlüsseln zur Verfügung gestellt. Diese Funktionalität benötigt das Kartenschlüssel-Management. Aufgrund der kartenabhängigkeit wird die Schlüsselableitung im `IMAFCAutCardProxy` bereitgestellt.

Der `IMAFCAppletManagerProxy` kümmert sich um die Ausführung von Kommandos mit höherer Management–Funktionalität. Dazu fasst er verschiedene Kommandos des `IMAFCVOPCommandProxy` zusammen. Der `IMAFCAppletManagerProxy` bildet die Schnittstelle zur Durchführung der Anwendungsfälle des Karten-Halters.

Eine Beschreibung der kartenspezifischen Realisierung der Schnittstellen des Multi-Applikations-Infrastruktur-Providers befindet sich im Kapitel Implementierung.

Abbildung 6.3 definiert Strukturen zur Verwendung mit den Schnittstellen des Multi-Applikations-Infrastruktur-Anbieters.

Die erste Schnittstelle ist `IMAFCAppletManagerResult`, welche einen generischen Rückgabetyp für Operationen der Schnittstelle `IMAFCAppletManagerProxy` bietet. Jeder initiale Aufruf einer der Methoden `list()`, `load()`, `install()`, `delete()` sowie `getFreeMemory()` des `IMAFCAppletManagerProxy` kann weitere APDU's zur

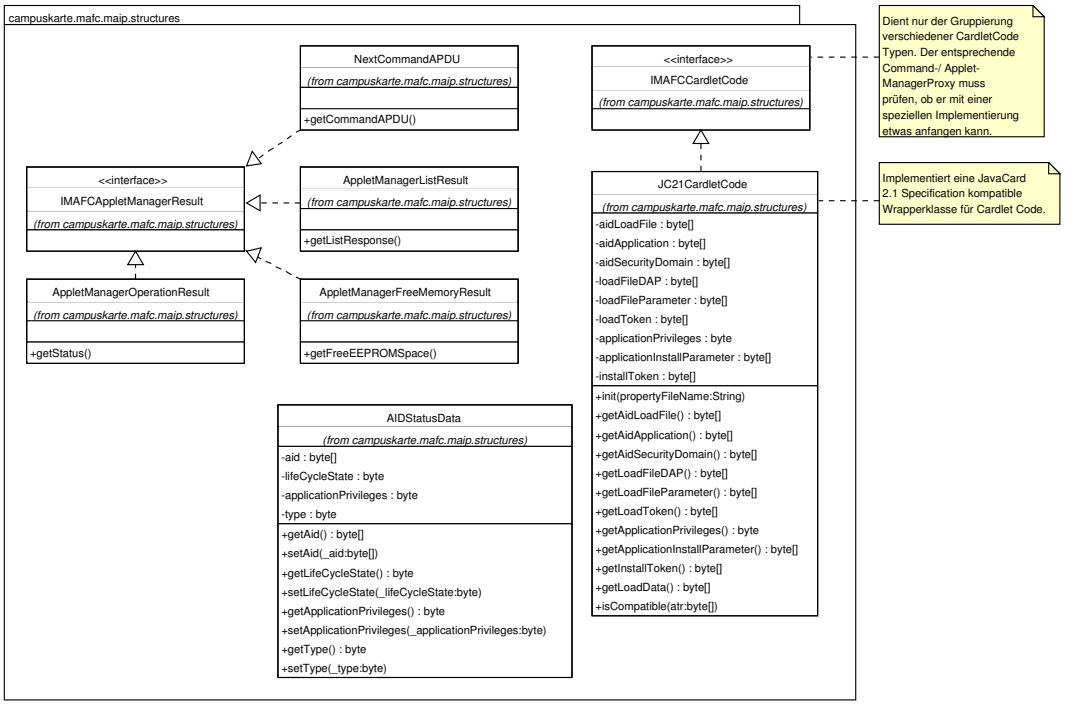

Abbildung 6.3: Datenstrukturen für den Multi-Applikations-Infrastruktur-Anbieter

Übertragung an die Chipkarte zurückgeben (Klasse `NextCommandAPDU`). Die Antwort-APDU der Chipkarte ist der gleichen Methode durch erneutes Aufrufen als Parameter zu übergeben. Es folgen entweder weitere `NextCommandAPDU`-Sequenzen oder die Operation wird mit einem Ergebnis beendet (`AppletManagerOperationResult`, `AppletManagerListResult`, `AppletManagerFreeMemoryResult`). Fehler werden, wie in der gesamten Implementierung, durch Ausnahmen behandelt (siehe Kapitel 7.2). Die Schnittstelle des `IMAFCAppletManagerProxy` muss wegen der Client-Server Architektur so aufwendig gestaltet werden. Eine `IMAFCAppletManagerProxy`-Implementierung kann nicht selbstständig Kommandos an die Chipkarte schicken, sondern muss diese einzeln zurückgeben.

Weiterhin gibt es die Klasse `AIDStatusData`. Diese nimmt Informationen auf, welche eine Chipkarte nach VOP 2.0.1 beim Abfragen des Karteninhaltes zurückgibt. Die Struktur nimmt jeweils einen Eintrag auf, eine Liste wird z.B. in `AppletManagerListResult` gebildet.

Eine weitere Schnittstelle, `IMAFCCardletCode` dient der Typisierung von Cardlets zum Laden auf die Chipkarte. Für diese Arbeit war es notwendig, eine Java Card 2.1 [43] kompatible Struktur zur Aufnahme von Cardlets bereitzustellen (`JC21CardletCode`). Da sich die einzelnen Elemente nicht verallgemeinern lassen, muss jede Klasse, welche `IMAFCCardletCode` benutzt, prüfen ob sie mit einer Spezialisierung etwas anfangen kann.

6.2.3 Karten–Status–Abfrage–System

Das Karten–Status–Abfrage–System[4] definiert Schnittstellen für die folgenden Anwendungsfälle. Diese sind in Abbildung 6.4 dargestellt.

[4] in der Implementierung Card Status Authority, kurz CSA genannt

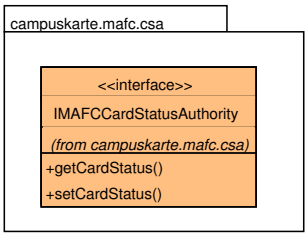

Abbildung 6.4: Schnittstellen für das Karten–Status–Abfrage–System

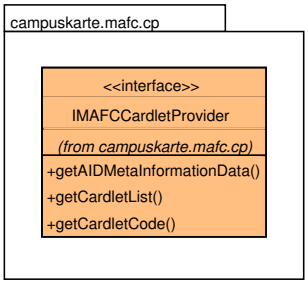

Abbildung 6.5: Schnittstellen für den Cardlet-Anbieter

- **Kartenstatus abfragen** - Wird definiert durch die Schnittstelle `IMAFCCardStatusAuthority`.
- **Kartenstatus verwalten** - Wird ebenfalls definiert durch die Schnittstelle `IMAFCCardStatusAuthority`.

Die Schnittstelle `IMAFCCardStatusAuthority` definiert Funktionalitäten zum Setzen und Abfragen von Kartenstatus-Zuständen. Diese sind in Kapitel 4.3.1 ausführlich beschrieben.

6.2.4 Cardlet–Anbieter

Der Cardlet–Anbieter[5] definiert die in Abbildung 6.5 dargestellten Schnittstellen. Diese basieren auf den folgenden Anwendungsfällen.

- **Cardlet bereitstellen** - Wird definiert durch die Schnittstelle `IMAFCCardletProvider`.
- **Cardlet Meta-Informationen bereitstellen** - Wird ebenfalls definiert durch die Schnittstelle `IMAFCCardletProvider`.

Die Schnittstelle `IMAFCCardletProvider` stellt drei wesentliche Funktionalitäten zum Cardlet-Management zur Verfügung. Mit der Funktionalität `getAIDMetaInformationData()` ist es möglich, Meta–Informationen zu bestimmten Cardlets auf der Chipkarte zu erhalten. Diese umfassen je nach Typ z.B. den ausgeschriebenen Namen, die Größe oder die digitale Signatur des Cardlets. Die möglichen Typen sind in Abbildung 6.6 enthalten. Mittels der Funktionalität `getCardletList()` können zur Installation auf einer Chipkarte verfügbare Cardlets ermittelt werden. Die Funktionalität `getCardletCode()` stellt ein Cardlet mit allen benötigten Informationen zur Installation auf einer Chipkarte bereit.

[5] in der Implementierung Cardlet Provider, kurz CP genannt

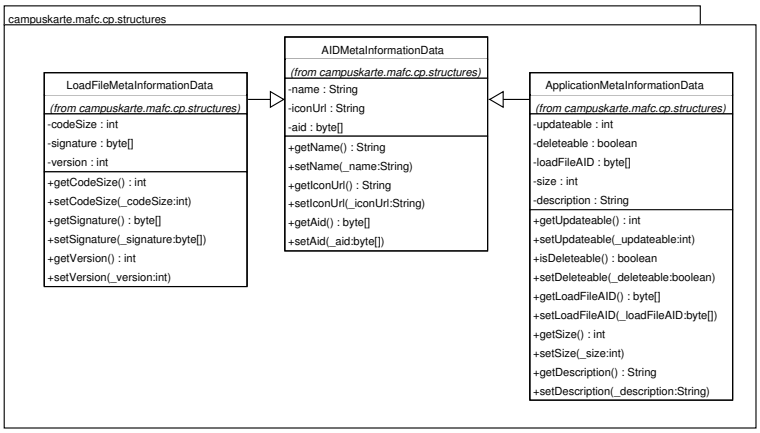

Abbildung 6.6: Datenstrukturen für den Cardlet–Anbieter

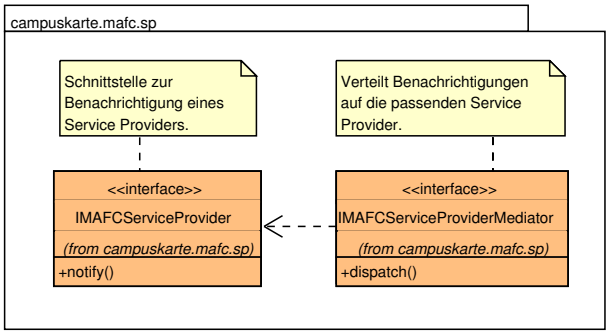

Abbildung 6.7: Schnittstellen für den Anwendungs–Betreiber

Abbildung 6.6 definiert Datenstrukturen zur Verwendung mit dem Cardlet-Anbieter. Die Klasse `AIDMetaInformationData` definiert einen Rückgabetypen für die Funktionalität `getAIDMetaInformationData()` der `IMAFCCardletProvider`-Schnittstelle. Konkrete Untertypen sind `LoadFileMetaInformationData` für Loadfiles sowie `ApplicationMetaInformationData` für Applications auf der Chipkarte. Die Bezeichnung Loadfile sowie Application entstammt der VOP 2.0.1 Spezifikation. Loadfiles sind Datendateien auf der Chipkarte, Applications hingegen ausführbare Anwendungen. Ein Cardlet umfasst immer mindestens ein Loadfile sowie eine Application.

6.2.5 Anwendungs–Betreiber

Der Anwendungs–Betreiber[6] definiert Schnittstellen für die folgenden Anwendungsfälle. Diese sind in Abbildung 6.7 dargestellt.

- **Anwendungs-Betreiber-Benachrichtigung** - Wird definiert durch die Schnittstelle `IMAFCServiceProvider` sowie `IMAFCServiceProviderMediator`.

Die Schnittstelle `IMAFCServiceProvider` stellt die Funktionalität zur Benachrichtigung eines Anwendungs-Betreibers nach erfolgreicher (De)-Installation eines Cardlets zur Verfügung. Er kann daraufhin weitere Maßnahmen einleiten. Da verschiedene Cardlets von verschiedenen Anwendungs-Betreibern angeboten werden

[6] in der Implementierung Service Provider, kurz SP genannt

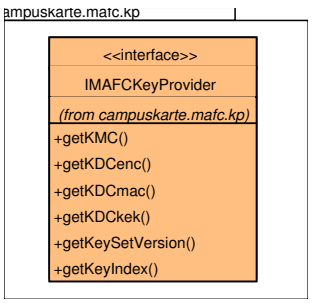

Abbildung 6.8: Schnittstellen für das Kartenschlüssel–Management

können, muss zu einem Cardlet der korrekte Anwendungs-Betreiber ermittelt werden. Diese Aufgabe übernimmt die Schnittstelle `IMAFCServiceProviderMediator` mit der Bereitstellung der Verteilungsfunktionalität `dispatch()`.

6.2.6 Kartenschlüssel–Management

Das Kartenschlüssel-Management[7] definiert Schnittstellen für die folgenden Anwendungsfälle. Diese sind in Abbildung 6.8 dargestellt.

- **Kartenschlüssel bereitstellen** - Wird definiert durch die Schnittstelle `IMAFC-KeyProvider`.

Die Schnittstelle `IMAFCKeyProvider` stellt Funktionalitäten zur Bereitstellung verschiedener Schlüssel für Chipkarten zur Verfügung. Nähere Informationen befinden sich in Abschnitt 6.6.3.

6.2.7 Dienste-Sicherheit

In der prototypischen Implementierung sind keine Sicherheitsfunktionen in den Diensten vorgesehen. Da diese im geschützten Bereich auf dem Server laufen, kann bei korrekter Programmierung und Absicherung des Servers kein Zugriff von außerhalb erfolgen. Angriffe von innen können jedoch nicht abgewehrt werden. Ist der Zugriff auf den Service–Handler gegeben, so können alle Dienste uneingeschränkt genutzt werden und z.B. die Schlüssel aller registrierten Chipkarten abgefragt werden. Für eine zukünftige Erweiterung würde sich ein Rechte-Management anbieten. Dieses könnte ein eigener Dienst sein. Beim Aufruf einer jeden Methode eines Dienstes sollte ein Sicherheits–Objekt übergeben werden, anhand welchem der Rechte–Management–Dienst entscheidet, ob die Funktionalität erlaubt oder gesperrt ist. Das Sicherheits–Objekt sollte sowohl Personen, als auch Rollen beschreiben können. Ansätze dazu finden sich z.B. in [50].

6.3 Server

Der Server bietet die Anwendungsfälle des Karten-Halters an. Dazu bedient er sich aller beschriebenen Dienste. Er kommuniziert mit einem Applet, welches auf dem Client laufen muss. Näheres dazu im Abschnitt 6.4 sowie 6.6.

[7] in der Implementierung Key Provider, kurz KP genannt

Zum Verständnis der folgenden Teile dieser Arbeit ist eine Erläuterung der verwendeten Definition eines Zustandes (engl. State) erforderlich. Dieser ist in der UML wie folgt definiert [33, Abschn. 3.75.1]:

„A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.“

Wenn im Folgendem von Zuständen die Rede ist, so bezieht sich dies auf die obige Definition. So kann es z.B. ein Objekt geben, welches den Zustand Karteninhalt anzeigen repräsentiert. Zusätzlich enthält dieses Objekt jedoch auch eine Methode, welche zum Anzeigen des Karteninhaltes notwendige Operationen ausführt (In den Diagrammen als Do gekennzeichnet). Weiterhin gibt es Entry und Exit Aktionen, welche jeweils beim Erreichen oder Verlassen des Zustandes ausgeführt werden. Ein Objekt, das einen Zustand nach der obigen Beschreibung repräsentiert, wird im folgenden Zustandsobjekt genannt.

- **Kartenauthentisierung** - Der Anwendungsfall Kartenauthentisierung sowie die enthaltenen Anwendungsfälle Kartenerkennung und Kartenstatus abfragen werden bereits durch den Multi–Applikations–Infrastruktur–Anbieter abgedeckt. Der Server muss jedoch eine Schnittstelle dieser Anwendungsfälle zum Benutzer anbieten. Dies geschieht durch die Server–Zustände `CardDetectionState`, `SelectCardManagerState`, `InitializeUpdateState`, `ExternalAuthenticateState` sowie `CheckAuthenticationState`. Diese basieren auf dem Aktivitätsdiagramm aus Abbildung 5.6.
- **Karteninhalt anzeigen** - Der Karteninhalt wird durch das Server-Zustand `RequestListState` dargestellt.
- **Verfügbare Cardlets anzeigen** - Die verfügbaren Cardlets werden durch den Server-Zustand `RequestAvailableCardletsState` dargestellt. Dieser Zustand basiert auf dem Aktivitätsdiagramm aus Abbildung 5.3.
- **Cardlet installieren** - Neue Cardlets werden über den Server-Zustand `RequestInstallState` installiert. Auch dieser Zustand basiert auf dem Aktivitätsdiagramm aus Abbildung 5.3.
- **Cardlet löschen** - Cardlets werden über die Server-Zustände `RequestConfirmDeleteState` sowie `RequestDeleteState` gelöscht. Dieser Zustand basiert auf dem Aktivitätsdiagramm aus Abbildung 5.4.

6.3.1 Kartenauthentisierung

Der Server funktioniert zustandsbasiert. Für jede neue Sitzung gibt einen Initialzustand Warte, in welchem auf neue Anfragen gewartet wird. Die erste erlaubte Operation ist die Kartenauthentisierung. Diese ist in Abbildung 6.9 dargestellt und basiert auf dem Anwendungsfall Kartenauthentisierung.

Der erste dargestellte Zustand, `RequestSession` ist logisch korrekt, wird jedoch in der prototypischen Implementation nicht als Zustand betrachtet, da die Session-Erzeugung vom Web-Server gesteuert werden kann (weitere Erläuterungen befinden sich im Kapitel 7.5).

Der zweite Zustand, `CardDetectionState` nimmt als Eingabe den ATR der eingelegte Chipkarte entgegen und führt die Kartenerkennung des Multi-Applikations-Infrastruktur-Anbieters durch. Anschließend wird ein kartentypspezifisches Kommando zum Auswählen des Card-Managers erzeugt und zurückgegeben.

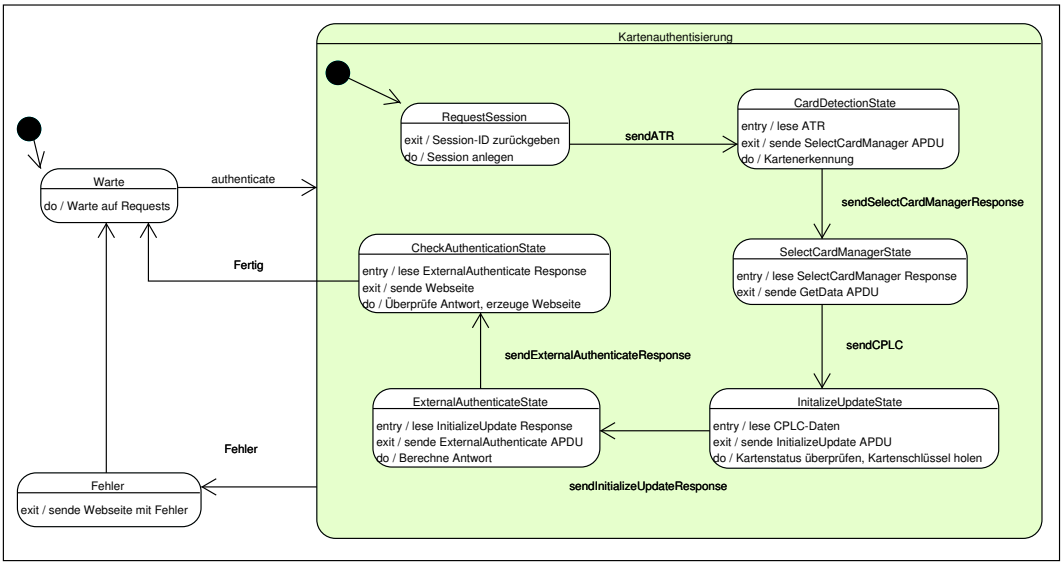

Abbildung 6.9: Zustandsdiagramm für die Kartenauthentisierung des Servers

Im dritten Zustand, `SelectCardManagerState` wird zunächst ausgewertet, ob der Card-Manager korrekt selektiert wurde. Anschließend wird ein kartentypspezifisches Kommando zum Abfragen der Seriennummer der Chipkarte zurückgegeben.

Der vierte Zustand, `InitializeUpdateState` liest die Antwortdaten des vorhergehenden Kommandos ein und berechnet daraus die CUID der eingelegten Chipkarte. Mit Hilfe der CUID kann der Kartenstatus über das Karten-Status-Abfrage-System ermittelt werden. Ist die Rekonfiguration der Chipkarte erlaubt, so werden die kartenspezifischen Schlüssel über das Kartenschlüssel–Management ermittelt. Mit diesen kann der `IMAFCAutCardProxy`–Dienst des Multi–Applikations–Infrastruktur–Anbieters aufgerufen werden, um für diesen, und die abschließenden Anwendungsfälle, Kommandos zur Authentisierung des Card-Managers zu erhalten. Zuerst wird nach VOP 2.0.1 ein sogenanntes InitalizeUpdate–Kommando erzeugt. Eine genauere Beschreibung befindet sich in der Protokollanalyse in Abschnitt 6.6.3.

Im fünften Zustand, `ExternalAuthenticateState` wird die Antwort des InitializeUpdate–Kommandos ausgewertet. Dabei wird der Card-Manager der Chipkarte authentisiert. Anschließend wird eine Antwort, das ExternalAuthenticate–Kommando erzeugt.

Im sechsten Zustand, `CheckAuthenticationState` wird die Antwort des ExternalAuthenticate–Kommandos überprüft. Ist dies erfolgreich, so ist auch der Server gegenüber der Chipkarte authentisiert und es existiert ein sicherer Kommunikationskanal. Es wird eine Webseite für den Benutzer erzeugt, welche die erfolgreiche Authentisierung anzeigt.

Sollte ein Fehlerzustand erreicht werden, so kann der Zustand Kartenauthentisierung jederzeit verlassen werden. In diesem Fall wird durch den server–internen Zustand Fehler eine Webseite für den Benutzer erzeugt, welche den Fehler anzeigt. Der einzige danach gültige Zustand ist eine erneute Kartenauthentisierung.

6.3.2 Karteninhalt anzeigen, Cardlets installieren und löschen

Nach einer erfolgreichen Authentisierung können drei neue Zustände des Servers erreicht werden. Diese sind in Abbildung 6.10 dargestellt.

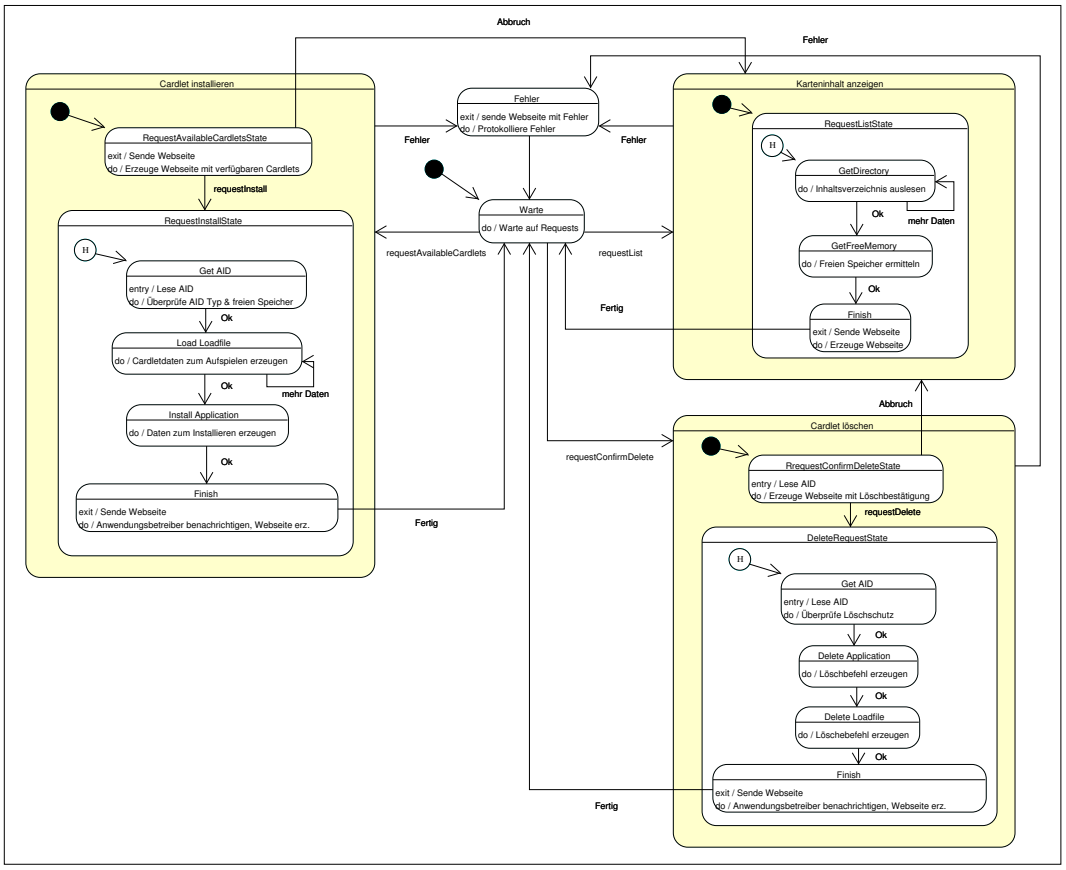

Abbildung 6.10: Zustandsdiagramm für die Karten-Halter Anwendungsfälle des Servers

Der Zustand `RequestListState` basiert auf dem Karten-Halter Anwendungsfall Karteninhalt anzeigen. Dieser Zustand besitzt weitere interne Zustände, welche verschiedene benötigte Kommandos für die Chipkarte erzeugen. Diese rufen Funktionalitäten der `IMAFCAppletManagerProxy`–Schnittstelle des Multi–Applikations–Infrastruktur–Anbieters auf. Wenn keine weitere Kommandos zu erzeugen sind, wird eine Webseite mit dem für den Karten-Halter relevanten Karteninhalt zurückgegeben. Der Server sichert die aktuelle Fassung des Karteninhaltes zur späteren Verwendung intern.

Der Zustand `RequestAvailableCardletsState` basiert auf dem Karten-Halter Anwendungsfall Verfügbare Cardlets anzeigen. Dieser Zustand kann rein server–intern durch Nutzung der Dienste des Cardlet–Providers ausgewertet werden. Falls vorhanden, so wird auf den ausgelesenen Karteninhalt zurückgegriffen, um die Anzeige bereits installierter Cardlets zu unterdrücken. Weiterhin werden nur Cardlets angezeigt, welche mit dem während der Kartenerkennung erkannten Kartentyp kompatibel sind. Es wird eine Webseite mit einer Liste der verfügbaren Cardlets zurückgegeben.

Der Zustand `RequestInstallState` basiert auf dem Karten-Halter Anwendungsfall Cardlet installieren. Dieser Zustand besitzt weitere interne Zustände, welche verschiedene benötigte Kommandos für die Chipkarte erzeugen. Diese benutzen Funktionalitäten der `IMAFCAppletManagerProxy`–Schnittstelle des Multi–Applikations–Infrastruktur–Anbieters. Nach erfolgreicher Installation eines Cardlets wird die `IMAFCServiceProviderMediator`–Schnittstelle der Anwendungs–Betreiber aufgeru-

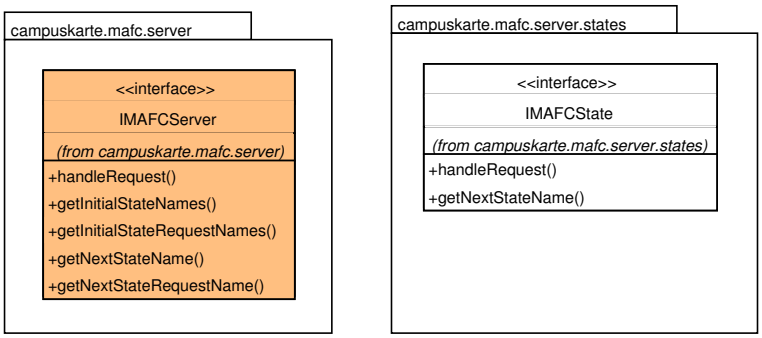

Abbildung 6.11: Schnittstellen für den Server

fen, um eine Nachricht an den entsprechenden Betreiber zu senden. Die erfolgreiche Installation wird protokolliert. Zum Abschluss wird eine Webseite mit einer erfolgreichen Installationsmeldung ausgegeben.

Der Zustand `RequestConfirmDeleteState` dient der Bestätigung, ob ein Cardlet wirklich gelöscht werden soll. Er gibt lediglich eine Webseite mit einem Warnhinweis zurück.

Der letzte mögliche Zustand, `DeleteRequestState` basiert auf dem Karten-Halter Anwendungsfall Cardlet löschen. Dieser Zustand besitzt weitere interne Zustände, welche verschiedene benötigte Kommandos für die Chipkarte erzeugen. Diese rufen Funktionalitäten der `IMAFCAppletManagerProxy`–Schnittstelle des Multi–Applikations–Infrastruktur–Anbieters auf. Die Generierung eines Löschbefehls findet nur statt, wenn das Cardlet nicht löschgeschützt ist. Nach erfolgreichem Löschen eines Cardlets wird die `IMAFCServiceProviderMediator`–Schnittstelle der Anwendungs–Betreiber aufgerufen, um eine Nachricht an den entsprechenden Betreiber zu senden. Das erfolgreiche Löschen wird protokolliert.

Beim Auftreten eines Fehlerfalles wird der server-interne Zustand Fehler aufgerufen. Dieser protokolliert den aufgetretenen Fehler und gibt eine Webseite mit einer Meldung zurück.

6.3.3 Schnittstellen für den Server

Um die hier beschriebenen Zustände des Servers möglichst einfach und generisch implementieren zu können, werden diese auf Zustandsobjekte abgebildet. Diese müssen eine Schnittstelle enthalten, über welche sie konform aufrufbar sind. Der Name dieser Schnittstelle ist `IMAFCState`, dargestellt ist sie in Abbildung 6.11. `IMAFCState` stellt eine Funktionalität zum Ausführen des aktuellen Zustandsobjektes zur Verfügung, sowie eine weitere zum Abfragen des logisch nächstens Zustandsobjektes. Dieses kann sich selbstverständlich während der Laufzeit des Servers ändern, z.B. durch eine Auswahl des Benutzers oder aus server–internen Gründen. Um eine komplexe Operation, wie die Kartenauthentisierung, durchzuführen, genügt somit die Kenntnis des initialen Zustandsobjektes.

Die Schnittstelle `IMAFCServer` definiert einen generischen Server, der verschiedene Zustände besitzen kann. Diese Schnittstelle ist auch in Abbildung 6.11 enthalten. Die Funktionalität `handleRequest()` führt das aktuelle Zustandsobjekt des Servers aus und setzt diesen danach auf das logisch Nächste. Mit `getInitialStateNames()` lassen sich die Namen aller initialen Zustände des Server ermitteln. `GetInitialStateRequestNames()` gibt die entsprechenden Anforderungen zum Aufruf der in-

itialen Zustandsobjekte zurück. Die Unterteilung zwischen Namen und Anforderungen wurden aus Sicht einer möglichen Integration in eine Benutzerschnittstelle für notwendig erachtet. So könnte ein Client die Liste der verfügbaren Zustände abrufen und diese dem Benutzer als Liste der einzelnen Namen zurückgeben. Diese können auch Beschreibungen oder Ähnliches enthalten. Eventuell sind die Namen verschiedener Zustände sogar gleich. Die Anforderung für ein bestimmtes Zustandsobjekt muss jedoch eindeutig definiert sein. Die Funktionalitäten `getNextStateName()` sowie `getNextStateRequestName()` geben dementsprechend den Namen oder die Anforderung für das Zustandsobjekt des logisch nächsten Zustandes zurück.

6.3.4 Server-Sicherheit

Die einzelnen Zustandsobjekte des Servers werden nicht über ein Rechte–Management verwaltet. Dies ist jedoch eine mögliche Option für zukünftige Erweiterungen. Stattdessen muss der Server ein Sitzungs-Management beherrschen, welches einen Benutzer einer Sitzung zuordnet. Jede Sitzung hat eine eigene Instanz des Service–Handlers, so dass andere Sitzungen nicht auf die Dienste fremder Sitzungen zugreifen können. Dies ist besonders relevant für sitzungsabhängige Daten. Bei korrekter Implementierung des Sitzungs-Managements sowie des Web-Servers sollten Vertraulichkeit und Integrität gegeben sein. Weiterhin müssen alle Zustände entsprechende Zusicherungen überprüfen, d.h. es muss geprüft werden, ob die gewünschte Operation erlaubt ist. So muss sich z.B. das Zustandsobjekt `RequestDeleteState` während der Ausführung überzeugen, dass das zu löschende Cardlet nicht löschgeschützt ist.

6.4 Client

Der Client dient dem Karten-Halter als Benutzungsschnittstelle zum Server. Er soll als Java-Applet innerhalb eines Web–Browser laufen. Die dazu benötigte Webseite soll der Server bereitstellen. Der Client funktioniert ohne ein Wissen über die dargestellten Inhalte. Er empfängt vom Server entweder Kommandos für die Chipkarte, welche er an diese weiterleitet oder Webseiten, die er anzeigt.

In Abbildung 6.12 ist das Aktivitätsdiagramm des Client dargestellt. Zum Start wird der Benutzer aufgefordert, einen Kartenleser auszuwählen, so es mehr als einen gibt. Anschließend wird das Open Card Framework initialisiert und auf das Einlegen einer Chipkarte gewartet. Sobald eine Chipkarte eingelegt wurde, wird eine neue Sitzung beim Server beantragt. Ist dies erfolgreich, so wird das Zustandsobjekt `CardDetectionState` des Servers aufgerufen. Dies erfolgt über eine spezielle URL, welche der Client beim Aufbau der Sitzung erhält. Die Aktivität Kartenauthentisierung startet eine Anfrage an den Server. Alle Anfragen laufen nach dem folgenden Schema ab:

1. Sende (nächste) Anfrage an Server.
2. Werte Antwort vom Server aus.
3. Wenn Antwort
   (a) Kommandos, dann sende diese an die Chipkarte. Gehe wieder zu 1.
   (b) Fehlermeldung, dann gib diese aus und warte auf Kartenentnahme. Gehe zu 4.

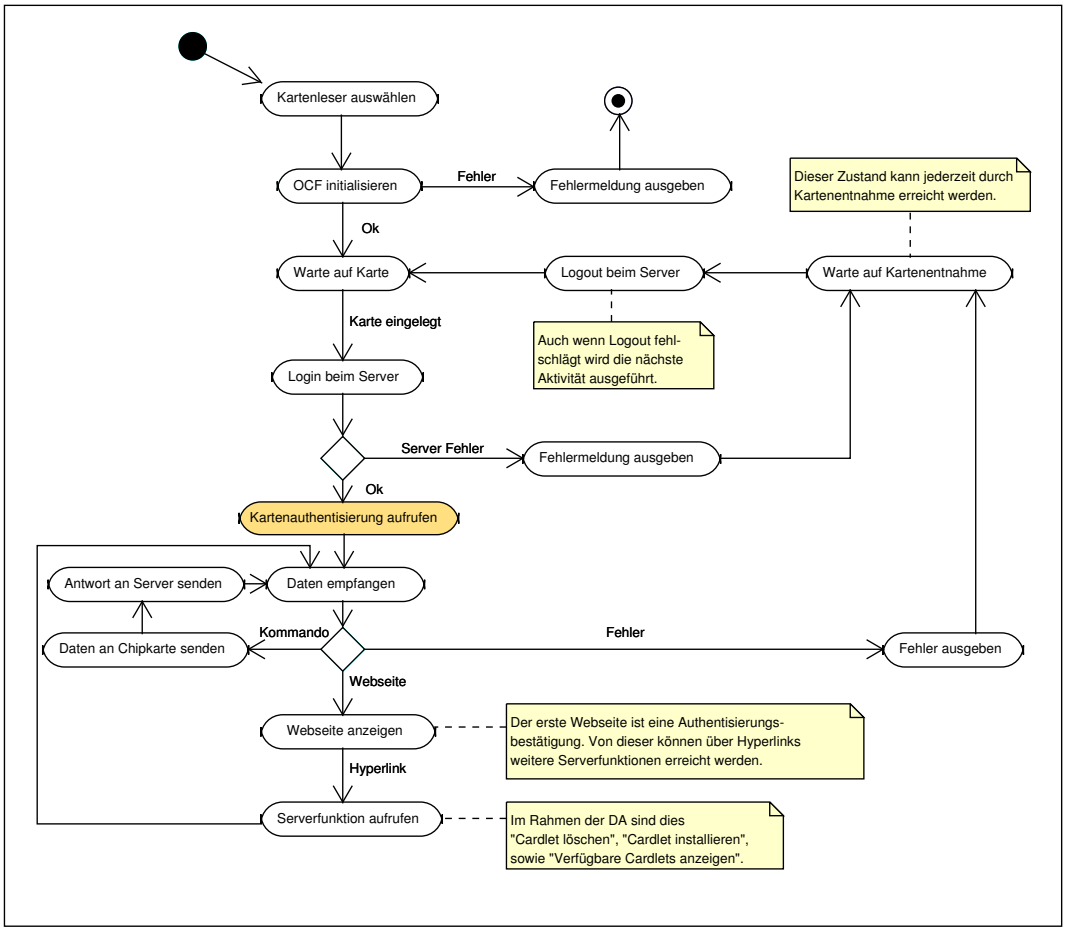

Abbildung 6.12: Aktivitätsdiagramm für den Client

(c) Webseite, dann gib diese aus und warte auf Hyperlink. Gehe zu 1.

4. Warte auf Kartenentnahme.

Die Aktivität Kartenentnahme kann jederzeit durch Entnahme der Chipkarte aufgerufen werden. Danach erfolgt eine Abmeldung vom Server. Es wird auf die Eingabe einer neuen Chipkarte gewartet.

Zur Realisierung der oben beschriebenen Funktionalität wurden zwei Schnittstellen definiert. Diese sind in Abbildung 6.13 dargestellt.

Die Schnittstelle `IMAFCClient` stellt Funktionalitäten zum Senden und Auswerten von Anfragen an den Server bereit. Um neue Kommandos zum Senden an die Chipkarte als mögliche Antwort des Servers auszuwerten, wird die Schnittstelle `IMAFCAppletProxy` benötigt. Die wichtigste Funktionalität dieser Schnittstelle ist `sendCommandAPDU`, welche ein Kommando an die Chipkarte sendet. Weiterhin muss die Schnittstelle über das Einlegen oder Entnehmen einer Chipkarte informiert werden.

6.4.1 Client-Sicherheit

Auf die Sicherheit der Kommunikationsstrecke zwischen Server und Client wird in Abschnitt 6.6 eingegangen. Wenn der Client eine authentisierte Verbindung zum

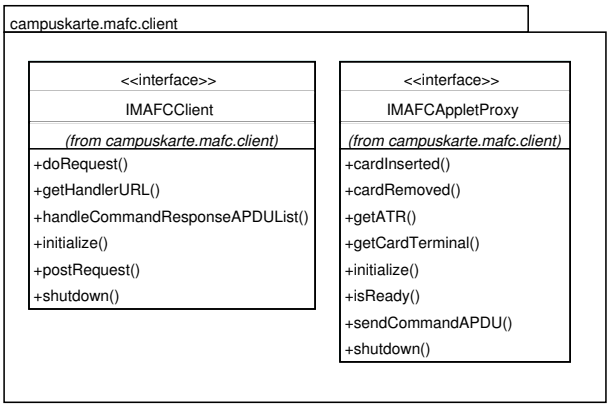

Abbildung 6.13: Schnittstellen für den Client

Server verwendet, so können ihm bei korrekter Implementierung keine falschen Antworten untergeschoben werden. Da der Client nur als Proxy zum Zugriff auf die Chipkarte sowie zum Anzeigen von Webseiten dient, ist unter Verwendung eines eigenen sicheren Kanals zur Kommunikation zwischen Server und Chipkarte keine Kompromittierung möglich. Allerdings unterstützen die in der prototypischen Implementierung verwendeten Chipkarten diesen nicht komplett, so dass der Client möglicherweise Antworten der Chipkarte fälschen kann, da diese nicht signiert oder verschlüsselt sind.

6.5 Chipkarte

Die im Campuskartenprojekt eingesetzten Java-Karten müssen im Rahmen dieser Arbeit nicht programmiert werden. Sie besitzen bereits eine spezielle Security-Domain, den Card–Manager. Dieser erlaubt die Bereitstellung eines sicheren Kanals zur Rekonfiguration der Chipkarte.

6.5.1 Chipkarten–Sicherheit

Die Sicherheit des Kommunikationskanals zwischen Chipkarte und Server bezüglich Vertraulichkeit, Integrität sowie Authentizität ist mit momentan verfügbaren Chipkarten nur teilweise zu erreichen.

Der Visa Open Platform Standard, welcher in der Version 2.0.1 von aktuellen Chipkarten unterstützt wird, sieht drei verschiedene Stufen für Security–Domains vor. Diese übernehmen teilweise die Funktionalität des Card–Managers.

- Die erste Stufe sieht die Bereitstellung und Überprüfung von Schlüsseln und Zertifikaten für andere Cardlets vor.
- Die zweite Stufe (Security Domains with mandated DAP verification or general DAP verification privilege), sieht zusätzlich die Überprüfung von geladenen Cardlet-Daten mittels eines Authentisierungspattern vor.
- Die dritte Stufe (Security Domains with Delegated Management), übernimmt das sichere Laden und Löschen von Applikation nach der Ausgabe der Karte.

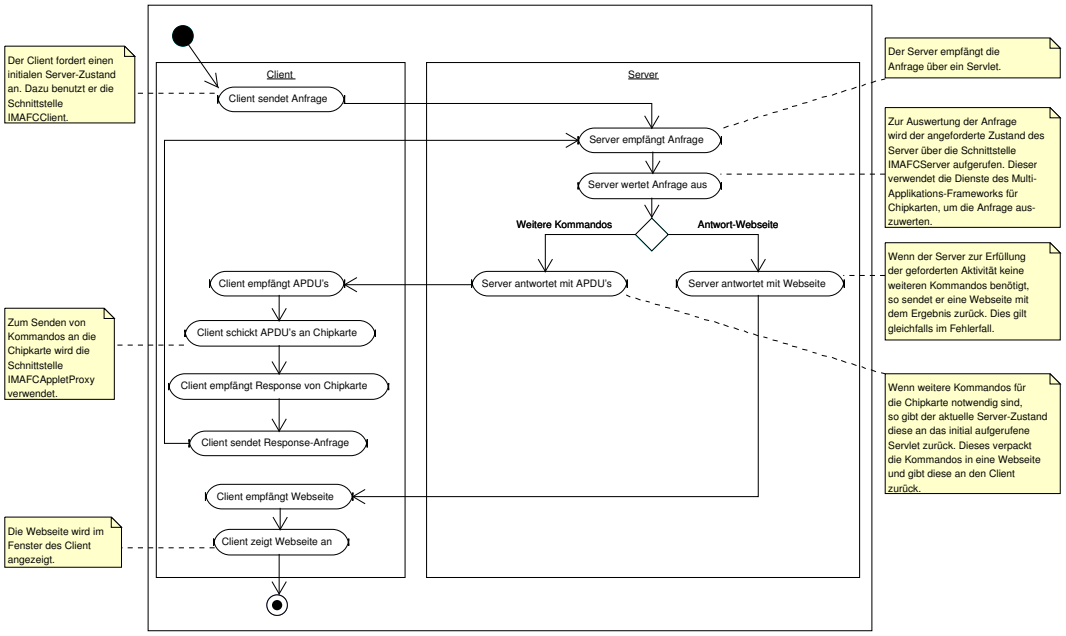

Abbildung 6.14: Aktivitätsdiagramm für die Kommunikation Client-Server

Aktuelle Java-Karten unterstützen lediglich die erste oder zweite Stufe. Allerdings ist die Überprüfung der geladenen Cardlet-Daten mittels eines Authentisierungspattern nicht zwingend erforderlich, da Cardlets auch über einen sicheren Kanal durch den Card–Manager übertragen werden können. Sollte dieser Kanal gebrochen werden und ein unlegitimiertes Cardlet auf die Chipkarte gelangen, so dürfte dieses, durch die Firewall auf der Java–Karte, welche verschiedene Cardlets voneinander trennt, keinen Zugriff auf sensible Daten erhalten. Zur Überprüfung der digitalen Signatur eines Cardlets müsste eine spezielle Security Domain zweiter Stufe programmiert werden. Dies wäre mit modernen Chipkarten technisch möglich, ist jedoch im Rahmen der Aufgabenstellung dieser Arbeit nicht erforderlich.

Ein wesentlicher Sicherheitsmangel, der teilweise technisch bedingt ist, besteht darin, dass der Header sowie die Rückgabe eines Kommandos des Card–Managers nicht verschlüsselt oder signiert erfolgt. Vertraulichkeit, Integrität sowie Authentizität können nur für den Datenteil eines Kommandos garantiert werden. Genauere Informationen befinden sich im Abschnitt 6.6.3.

6.6 Kommunikation Client-Server

Bei der Betrachtung der Kommunikation umfasst der Client die Chipkarte und der Server die Dienste. Es wird in diesem Abschnitt somit nicht nur die Kommunikation zwischen Client-Anwendung und Web-Server, sondern auch zwischen der Chipkarte und den Diensten betrachtet.

6.6.1 Ablauf

Der grundlegende Kommunikationsverlauf zwischen Client und Server ist in Abbildung 6.14 dargestellt. Der Client sendet im ersten Schritt eine Anfrage mit einem initialen Server-Zustand als Parameter. Der Server empfängt die Anfrage und wertet

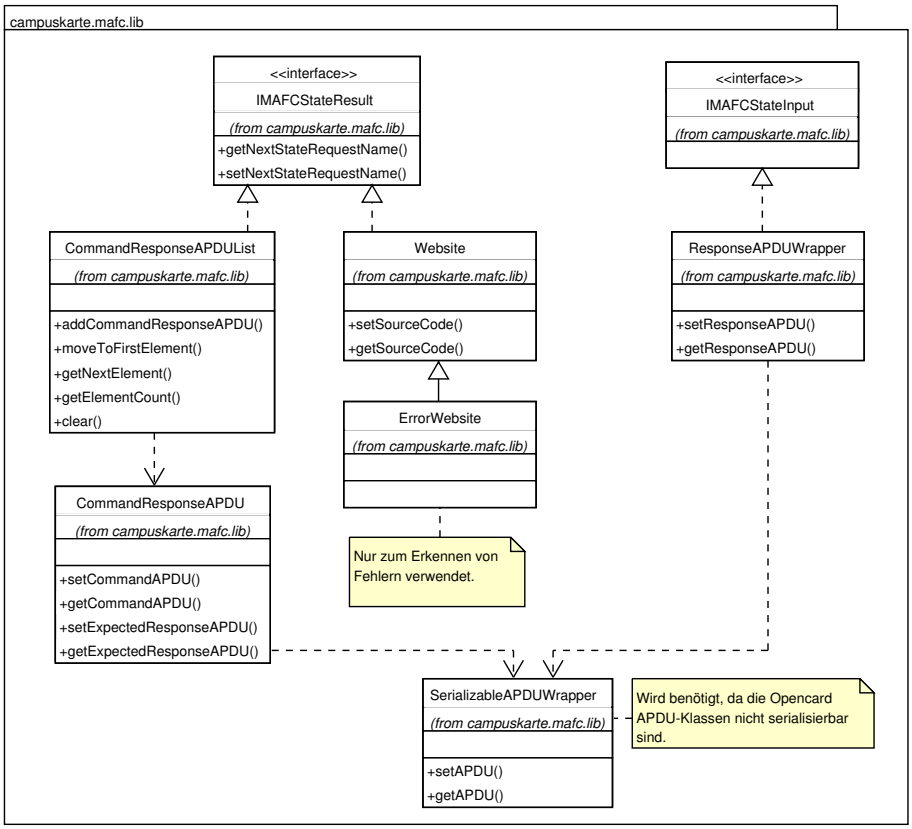

Abbildung 6.15: Datenstrukturen zur Client-Server Kommunikation

diese mit Hilfe der Dienste des Multi-Applikations-Frameworks für Chipkarten aus. Wird zur Bearbeitung der Anfrage Funktionalität von der Chipkarte benötigt, so werden von den Diensten entsprechende APDU-Kommandos erzeugt. Diese werden in eine Webseite verpackt an den Client zurückgegeben. Der Client erkennt den Typ der Antwort und schickt dementsprechend Anfragen an die Chipkarte. Antwortet der Server mit einer Antwort-Webseite, so ist die Operation entweder beendet oder ein Fehler aufgetreten. In beiden Fällen muss die Webseite dem Benutzer angezeigt werden.

Um den Client möglichst einfach halten zu können, soll dieser serialisierte Objekte vom Server erhalten. In der prototypischen Implementierung dienen diese lediglich als Datentypen, um ein aufwendiges Parsen von Eingabedaten zu verhindern. In späteren Erweiterungen kann jedoch auch Funktionalität in die Objekte gepackt werden. Eine Antwort des Clients an den Server wurde gleichfalls als serialisiertes Objekt spezifiziert. Da dies aber eine mögliche Sicherheitslücke darstellt, welche im folgenden Sicherheitsentwurf beschrieben wird, wurde in der prototypischen Implementierung darauf verzichtet.

Die verschiedenen Typen und Datenstrukturen sind in Abbildung 6.15 dargestellt. `IMAFCStateInput` dient als Typ für Rück- und Eingabewerte an den Server. Die einzige Implementation stellt der `ResponseAPDUWrapper` dar. Aus Sicherheitsgründen wird dieses Objekt jedoch nicht im Client, sondern erst im Server erzeugt. `IMAFCStateResult` stellt die Antwort des Servers auf eine Anfrage dar. Ein möglicher Rückgabetyp ist eine Liste mit Kommandos für die Chipkarte (`CommandResponseAPDUList`). Wenn die Operation vom Server abgeschlossen wurde, so wird als Antwort der Typ `Website` zurückgegeben. Eine Spezialisierung von `Website` ist

`ErrorWebsite`, welche keine neue Funktionalität bereitstellt, sondern lediglich zur Unterscheidung des Antworttyps dient.

6.6.2 Sicherheit

In diesem Abschnitt wird beschrieben, inwieweit die Schutzziele Vertraulichkeit, Integrität sowie Verbindlichkeit der Rolle Karten–Halter sichergestellt werden können. Dazu wird auch eine Analyse der verwendeten Protokolle durchgeführt.

Sicherheitsentwurf

Die Sicherung der Verbindung zwischen dem Client und Server soll über das HTTPS-Protokoll erfolgen (siehe Kapitel 3.5). Bei diesem Protokoll wird nach einer Authentizitätsprüfung des Servers die Vertraulichkeit und Integrität der Verbindung bidirektional über das Netzwerk sichergestellt.

Die Verbindung von der Chipkarte zum Server kann jedoch immer noch auf dem Client-PC kompromittiert werden. Dieser könnte manipulierte APDU–Sequenzen oder Antworten an die Chipkarte schicken. Um dies zu verhindern, sieht der VISA Open Platform Standard 2.0.1 verschiedene Sicherheitsmechanismen vor. Allerdings sind wesentliche Sicherheitskomponenten dieses Standards optional, so dass viele auf dem Markt erhältlich Chipkarten diese nicht oder nur eingeschränkt unterstützen.

Verpflichtend vorgesehen ist eine gegenseitige Authentisierung zwischen der Chipkarte und der Gegenstelle, im Falle dieser Arbeit sind dies die Chipkartendienste. Das verwendete Protokoll wird im Abschnitt 6.6.3 analysiert.

Nach einer erfolgreichen gegenseitigen Authentisierung verfügen Chipkarte und Server über Sitzungsschlüssel, welche sie zur sicheren Kommunikation verwenden können. Aus Kompatibilitätgründen kann jedoch nur der Datenteil eines Kommandos für die Chipkarte verschlüsselt und signiert werden. Eine Signatur der Antwort von der Chipkarte ist gänzlich optional. Genauere Betrachtungen folgen im Abschnitt 6.6.3.

Serialisierte Objekte

Besondere Vorsicht ist bei der Kommunikation mit serialisierten Objekten geboten. Diese ermöglicht einerseits eine schöne softwaretechnische Architektur, öffnet andererseits aber Sicherheitslücken. In einer ersten Version sendete und empfing der Server serialisierte Objekte. Diese dienten nur der Kapselung von Datenstrukturen, d.h. sie besaßen nur get- und set-Methoden. Für zukünftige Erweiterungen war es angedacht, dem Client auch den Code zum Bearbeiten der Daten mitzuschicken. Dies stellt insofern noch kein Problem dar, da der Server sich dem Benutzer gegenüber per HTTPS–Protokoll identifiziert und der Benutzer diesem zur Verwendung der Dienste vertrauen muss. Ein Dritter kann sich nicht mit trivialen Mitteln als zertifizierter Server ausgeben. Das Problem bestand jedoch darin, dass der Client in einer ersten Version auch serialisierte Objekte als Antwort an den Server schickte. Diese Objekte wurden auf ihren Typ geprüft und enthielten nur Daten. Allerdings besitzt jedes serialisierbare Objekt in Java einen parameterlosen Konstruktor, welcher bei der Deseriellisierung aufgerufen wird. In diesen Konstruktor kann ein Angreifer beliebigen Code einschleusen. Ganz so trivial ist es dennoch nicht, da Java Prüfsummen über die Klasse des Objektes bildet und mit der lokal (hier auf dem Server)

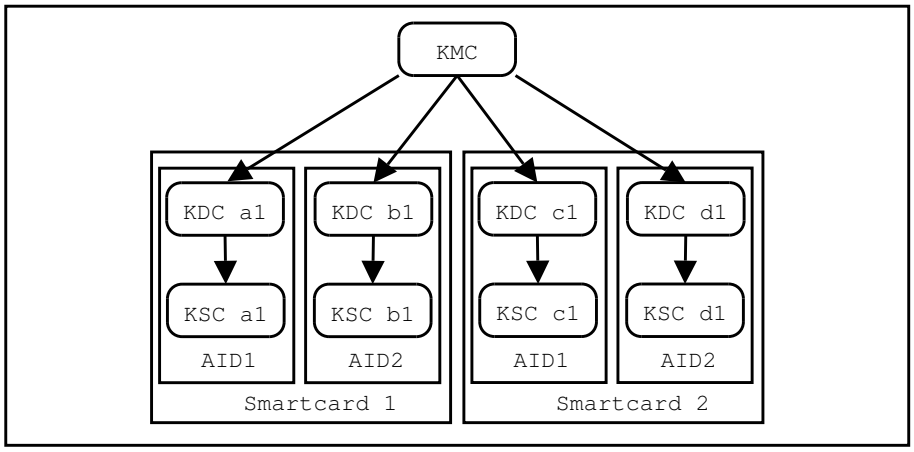

Abbildung 6.16: Schlüsselhierarchie für Java-Karten

vorhandenen Klasse vergleicht. Auch wenn ein praktischer Angriff somit schwierig wird, dürfte er dennoch nicht unmöglich sein. Aus Sicherheitsgründen sendet der Client seine Antworten in der aktuellen Fassung nur noch als reine Daten, die erst im Server zu einem Objekt zusammengesetzt werden.

6.6.3 Protokollanalyse

In diesem Abschnitt findet eine Beschreibung und Analyse der verwendeten Protokolle statt. Es wird der offizielle Visa Open Platform Standard 2.0.1 [19] betrachtet. Einige Hersteller verwenden proprietär abgeänderte Varianten zur Schlüsselerzeugung, der Ablauf des Protolles ist jedoch einheitlich.

Gegenseitige Authentisierung Anwendung und Java-Karte

Eine Java-Karte führt nach der Visa Open Platform Spezifikation [19, Abschn. 10.2] eine gegenseitige Authentisierung mit der Anwendung durch[8]. Die Authentisierung basiert auf der Überprüfung der gemeinsamen Kenntnis eines symmetrischen Schlüssels. Als Ergebnis können beide Seiten verschiedene Sitzungsschlüssel[9] (Session–Keys) berechnen, welche zur sicheren Kommunikation verwendet werden müssen.

Die verwendeten Schlüssel sind hierarchisch angeordnet, siehe Abbildung 6.16 [56, S. 36]. Es existiert ein Hauptschlüssel für jede gelieferte Chipkartencharge (KMC). Von diesem werden für jede Chipkarte und jede Security–Domain eindeutige Schlüssel abgeleitet (KDC). Dazu werden als eindeutiges Kartenmerkmal die Chip-Id, sowie die AID der angesprochenen Security–Domain verwendet. Die abgeleiteten Schlüssel dienen als Hauptschlüssel zur Verschlüsselung (KDC_{ENC}), zur Authentitätsprüfung (KDC_{MAC}) sowie zur sicheren Schlüsselübertragung (KDC_{KEK}). Im Zuge der gegenseitigen Authentisierung werden KDC_{ENC} und KDC_{MAC} benutzt, um Sitzungsschlüssel (KSC_{ENC}, KSC_{MAC}) zu berechnen. Die Herleitung der Schlüssel wird genau in [56, S. 54ff] beschrieben.

Im ersten Schritt der Authentisierung sendet die Anwendung ein Host–Challenge[10]

[8] Die Anwendung ist in dieser Arbeit der Chipkartendienst.
[9] Ein Sitzungsschlüssel ist ein symmetrischer Schlüssel, welcher nur für die aktuelle Sitzung gilt.
[10] Ein Challenge ist in diesem Zusammenhang eine Zufallszahl.

an die Java-Karte. Die Karte generiert daraufhin ein Card–Challenge und verknüpft dieses mit dem Host–Challenge sowie dem geheimen symmetrischen Schlüssel KDC_{ENC} zu verschiedenen Sitzungsschlüsseln. Mit einem dieser Sitzungsschlüssel wird ein Karten–Kryptogramm (Card Cryptogram) erzeugt. Das Karten–Kryptogramm wird zusammen mit der Card–Challenge und anderen Daten (Derivation–Data) zurückgegeben.

Die Anwendung verfügt jetzt über die selben Daten wie die Chipkarte und kann eine Kopie des Karten–Kryptogramms generieren. Stimmt dieses mit dem erhaltenen Karten–Kryptogramm überein, so ist die Karte authentisiert.

Die Anwendung generiert jetzt ein Host–Kryptogramm und sendet dieses an die Chipkarte zurück. Da die Chipkarte ebenfalls über alle Informationen verfügt, um eine Kopie des Host–Kryptogramms zu generieren, kann dieses mit dem übermittelten Host–Kryptogramm verglichen werden. Stimmen beide überein, so ist auch die Anwendung gegenüber der Karte authentisiert.

Zur anschließenden sicheren Kommunikation werden die Sitzungsschlüssel KSC_{ENC} und KSC_{MAC} verwendet.

Im Folgenden ist der Ablauf einer erfolgreichen Authentisierung beschrieben. Die verwendeten Bezeichner sind in Tabelle 6.2, die verwendeten Funktionen in Tabelle 6.3 enthalten. Das Plus–Zeichen (+) ist als Konkatenation zu verstehen, es setzt zwei Teilausdrücke zusammen.

1. $A : Generate(R_H)$
2. $A \Rightarrow J : Initialize\_Update(R_H)$
3. $J : Generate(R_C, C_C, KSC_{MAC}, KSC_{ENC})$
4. $J \Rightarrow A : Initialize\_UpdateResponse(D_{DER}, R_C, C_C)$
5. $A : Generate(C_H, KSC_{MAC}, KSC_{ENC})$
6. $A : Verify(C_C)$
7. $A \Rightarrow J : External\_Authenticate(C_H, MAC(KSC_{ENC}, C_H))$
8. $J : Verify(C_H)$
9. $J \Rightarrow A : External\_AuthenticateResponse(OK)$

Nach der erfolgreichen gegenseitigen Authentisierung können alle Visa Open Platform Operationen, wie z.B. Karteninhalt abfragen oder Cardlet löschen durchgeführt werden.

Vertraulichkeit und Verbindlichkeit der Kommunikation

Die Visa Open Platform Spezifikation 2.0.1 [19, Abschn. 12-1] sieht zur Verbindlichkeit der Kommunikation die Signatur von APDU's mittels einem symmetrischen Algorithmus vor. Um die Vertraulichkeit zur wahren, kann der Datenteil zusätzlich symmetrisch verschlüsselt werden. Nicht verschlüsselt oder signiert wird der Rückgabewert der Chipkarte. Dies ist zukünftigen Erweiterungen vorbehalten, da momentan erhältliche Chipkarten diese Funktionalität nicht unterstützen.[11]

[11] Selbst im Rahmen der Ausschreibung des Campuskartenprojektes der TU-Berlin konnte kein Anbieter eine solche Chipkarte liefern, obwohl diese Eigenschaften optional gefordert waren.

Bez.	Name	Berechnung/Beschreibung
A	Anwendung	Bezeichner für die Anwendung.
J	Java-Karte	Bezeichner für die Security-Domain auf der Java-Karte, welche authentisiert wird.
OK	Ok.	Konstante.
R_H	Host-Challenge	$R_H = R_{H1} + R_{H2}$ $(R_{H1} = RAN(), R_{H2} = RAN())$
R_C	Card-Challenge	$R_C = R_{C1} + R_{C2}$ $(R_{C1} = RAN(), R_{C2} = RAN())$
C_C	Karten-Kryptogramm	$C_C = MAC(KSC_{ENC}, R_H + R_C)$
C_H	Host-Kryptogramm	$C_H = MAC(KSC_{ENC}, R_C + R_H)$
D_{Der}	Derivation-Data	Kartenspezifisch, wird zur Schlüsselableitung für KDC_{ENC} und KDC_{MAC} benötigt.
KMC	Kartenausgeber-Hauptschlüssel	Statischer 3DES-Schlüssel.
KDC_{ENC}	Verschlüsselungs-Schlüssel	3DES-Schlüssel: Wird mittels D_{Der} sowie dem KMC abgeleitet, siehe [56, S. 54].
KDC_{MAC}	Authentikations-Schlüssel	3DES-Schlüssel: Wird mittels D_{Der} sowie dem KMC abgeleitet, siehe [56, S. 54].
KSC_{ENC}	Verschlüsselungs-Sitzungsschlüssel	3DES-Schlüssel: $KSC_{ENC} = ENC(KDC_{ENC}, R_{C2} + R_{H1}) + ENC(KDC_{ENC}, R_{C1} + R_{H2})$
KSC_{MAC}	Authentikations-Sitzungsschlüssel	3DES-Schlüssel: $KSC_{MAC} = ENC(KDC_{MAC}, R_{C2} + R_{H1}) + ENC(KDC_{MAC}, R_{C2} + R_{H2})$
$Apdu$	APDU	$Apdu = Apdu_{Header} Apdu_{Data}$
Mac	MAC einer APDU	Ergebnis der Funktion $MAC()$

Tabelle 6.2: Bezeichner für Java-Karten Authentisierung und Verschlüsselung

Zur symmetrischen Signatur der APDU wird diese zuerst mit einem Padding versehen. Dies ist erforderlich, da ein symmetrischer Blockalgorithmus verwendet wird. Die APDU mit Padding besitzt eine durch acht teilbare Länge. Jeweils acht Bytes dienen als Eingabe für einen 3DES/CBC Algorithmus zur Signatur mittels KDC_{Mac}. Der Initialisierungsvektor wurde während der gegenseitigen Authentisierung, bei der Signatur von C_H, gesetzt. Das Ergebnis einer Operation dient im nächsten Schritt als neuer Initialisierungsvektor. Das letzte Ergebnis bildet den Message Authentication Code, kurz MAC genannt. Dieser dient auch als Initialisierungsvektor für die nachfolgende Signatur.

Das Protokoll zur symmetrischen Signatur einer APDU nach VOP 2.0.1 ist im Folgendem dargestellt. Die verwendeten Bezeichner sind ebenfalls in Tabelle 6.2, die verwendeten Funktionen in Tabelle 6.3 enthalten.

1. $Apdu = PAD(Apdu)$
2. $Mac = MAC(KSC_{MAC}, Apdu)$

Funktion	**Beschreibung**
$Generate(x, ..)$	Erzeugt x nach Tabelle 6.2.
$Verify(x, ..)$	Verifiziert die Gültigkeit von x, indem ein x' erzeugt wird und auf Gleichheit zu x geprüft wird.
$Initialize\_Update(..)$	Beginnt die gegenseitige Authentisierung.
$Initialize\_UpdateResponse(..)$	Antwort auf $Initialize\_Update$.
$External\_Authenticate(...)$	Beendet die gegenseitige Authentisierung.
$External\_AuthenticateResponse(...)$	Antwort auf $External\_Authenticate$.
$ENC(key, data)$	Führt eine 3DES/CBC-Veschlüsselung von $data$ mittels key durch.
$MAC(key, data)$	Erzeugt einen 3DES/CBC-Authentikations-Code für $data$ mittels key.
$RAN()$	Erzeugt eine 4-Byte Zufallszahl.
$PAD(data)$	Fügt ein Padding zu $data$ hinzu.
$UNPAD(data)$	Entfernt ein Padding von $data$.

Tabelle 6.3: Funktionen für Java-Karten Authentisierung und Verschlüsselung

3. $Apdu = UNPAD(Apdu)$
4. $Apdu = Apdu + Mac$

Zur symmetrischen Verschlüsselung der APDU ist eine vorhergehende Signatur erforderlich. Der APDU-Header sowie der MAC werden abgetrennt und erst am Schluss der Operation wieder angefügt. Die verbleibende APDU wird gepaddet, wobei das Padding ein Teil der verschlüsselten Daten wird. Die Länge der APDU ändert sich entsprechend. Anschließend wird der Datenteil der APDU mittels 3DES/CBC Algorithmus verschlüsselt. Der Initialisierungsvektor wird dabei auf Null gesetzt. Zum Abschluss der Operation wird der ADPU-Header und der MAC wieder angefügt.

Das Protokoll hat folgenden Aufbau, die verwendeten Bezeichner und Funktionen entsprechen denen der vorhergegangenen Protokolle.

1. $Apdu_{Data} = PAD(Apdu_{Data})$
2. $ENC(KSC_{ENC}, Apdu_{Data})$
3. $Apdu = Apdu_{Header} + Apdu_{Data} + Mac$

Mögliche Angriffspunkte

Im Folgenden werden mögliche Angriffe beschrieben, welche sich aus der vorangegangenen Protokollanalyse ergeben haben.

- Die Qualität des Host-Challenge ist sehr wichtig. Zwar generiert auch die Chipkarte ein Card-Challenge, allerdings werden beide Challenges verknüpft, um die Sitzungsschlüssel zu berechnen.

- Durch Befragen der Chipkarte mit bewusst gewählten Host–Challenges können Daten zur Einschränkung des Suchbereichs ermittelt werden. Eventuell kann mit Hilfe der Host– und Card–Challenge daraus der KDC_{ENC} berechnet werden.

- Das Karten–Kryptogramm wird sehr früh zurückgegeben. Da dieses durch den Host verifiziert werden muss, kann hier ein Brute-Force Angriff zum Erhalten des KSC_{ENC} ansetzen, ohne eine weitere Kommunikation mit der Chipkarte durchzuführen.

- Einige Kartenhersteller limitieren den Aufruf des *Initialize_Update* Kommandos bis zur endgültigen Kartensperrung, so dass nur eine begrenzte Anzahl von möglichen Kombinationen aus Host- und Card-Challenge sowie Karten-Kryptogrammen abgerufen werden können. Dieser Zähler wird erst nach einer erfolgreichen Authentisierung zurückgesetzt. Allerdings ist der vorgenannte Angriff immer noch möglich.

- Da der Header einer APDU nicht verschlüsselt ist, kann keine vollständige Vertraulichkeit hergestellt werden. Ein Angreifer, welcher die Kommunikation auf dem Client mithören kann, erfährt, welche Kommandos aufgerufen werden. Eine Modifikation ist nicht möglich, da auch der Header in die Signatur einbezogen wurde.

- Der Rückgabewert einer Card-Manager Operation ist unverschlüsselt und unsigniert. Dies ist ein schwerwiegender Nachteil, da ein Angreifer den Status von Operationen beliebig fälschen kann, so z.B. den Karteninhalt oder die Bestätigung einer Löschoperation. In VOP 2.0.1 sind Erweiterungen vorgesehen, welche eine Signatur des Rückgabewertes vorsehen, allerdings wird dies momentan von keiner marktüblichen Chipkarte unterstützt.

- Sollte allerdings ein Chipkartenkommando abgefangen und durch eine gefälschte Antwort abgeschlossen werden, so fällt dies beim nächsten Zugriff auf die Chipkarte auf. Dies geschieht durch den Initialisierungsvektor zur Generierung des MAC, welcher auf dem Host aktualisiert wurde, auf der Chipkarte jedoch nicht. Dies nützt aber nichts, wenn auch alle folgenden Zugriffe auf die Chipkarte abgefangen werden.

- Ausgeschlossen ist die einfache Einschleusung fremden Codes auf die Chipkarte, da dieser in den Datenteilen der APDU's enthalten und verschlüsselt und signiert ist. Wie festgestellt, kann aber nicht garantiert werden, dass das Cardlet wirklich auf der Chipkarte installiert wurde. Dies kann nur der Anwendungs–Betreiber feststellen, indem er das Cardlet über eine sichere Funktion authentisiert.

Kapitel 7

Prototypische Implementierung

Das Kapitel Prototypische Implementierung beschreibt die Verfeinerung der Darstellungen aus Analyse und Entwurf hin zu einem lauffähigen Prototypen. Dabei werden nicht nur definierte Schnittstellen durch konkrete Klassen implementiert, sondern auch zur Funktion notwendige Erweiterungen vorgenommen.

Viele Details, welche in den vorangegangenen Kapiteln nicht betrachtet wurden, kommen erst bei der Implementierung zum Tragen. Dies bezieht sich z.B. auf die Fehlerbehandlung oder weitere Dienste, welche sehr systemspezifische Eigenschaften übernehmen. Weiterhin müssen durch Analyse und Entwurf vorgegebene Abläufe aus Zustands- und Aktivitätsdiagrammen berücksichtigt werden. Die Implementierung umfasst aber auch weitergehende Themengebiete, wie z.B. Quelltext-Versionsverwaltung oder Tests.

Eine wesentliche Konzentration der prototypischen Implementierung dieser Arbeit liegt auf den Anwendungsfällen des Karten–Halters. Für diesen wurde eine Benutzerschnittstelle bereits im Entwurf beschrieben. Zur praktischen Ausführung benötigen aber auch verschiedene andere Komponenten nicht nur Zugriffs-, sondern auch Konfigurationsschnittstellen. Diese sind aus Aufwandsgründen nicht im Entwurf berücksichtigt worden, in der prototypischen Implementierung jedoch unerlässlich. Ich habe versucht, sie so flexibel und einfach wie möglich zu halten.

In [49] wird im Kapitel „Why We Model“, unter der Überschrift „The Importance of Modeling“, eine viel verbreitete Herangehensweise zur Softwareentwicklung mit der klassischen Architektur verglichen: Je einfacher (und weniger komplex) das zu lösende Problem ist, je weniger Planung und Entwurf werden benötigt. Um eine Hundehütte zu bauen, braucht man nur einen Stapel Holz und Nägel sowie einfache Werkzeuge, wie einen Hammer und eine Säge. Wenn der Hund die Hütte nicht mag, kann man sich entweder einen neuen Hund anschaffen oder von vorn beginnen. Wenn man ein Haus für seine Familie bauen möchte, kann man die gleiche Herangehensweise verwenden.[1] Vermutlich werden die Frau und Kinder jedoch wesentlich höhere Ansprüche haben als der Hund. Da die Kosten und der Zeitaufwand ungleich höher liegen, zahlt sich ein wenig vorhergehende Planung schnell aus. Wenn man schließlich ein Hochhaus bauen will, so wird man dies kaum mit einem Stapel Holz und einem Hammer beginnen. Zumeist werden solche Häuser mit dem Geld anderer Leute gebaut, welche ganz spezielle Vorstellungen und Anforderungen an das Gebäude haben. Diese sich ständig ändernden Anforderungen müssen mit anderen Leuten kommuniziert und abgesprochen werden, da ein solch großes Projekt nur

[1] Aufgrund amtlicher Vorschriften ist dies in Deutschland nicht so einfach möglich.

durch erfolgreiche Zusammenarbeit realisiert werden kann.

Ich würde die Realisierung der Dienste der prototypischen Implementierung mit der zweiten Vorangehensweise vergleichen. Ich hatte nur begrenzte Kosten und keine externen Anforderungen. Es existieren lediglich vorgegebene Schnittstellen zum System, welche im Entwurf beschrieben wurden. Die Struktur der Dienste wurde aus verschiedenen Gründen nicht weiter entworfen.[2] Der wichtigste dieser Gründe sind die Kosten, in diesem Fall die Zeit. Das Ziel dieser Diplomarbeit ist eine Art Machbarkeitsstudie, welche demonstrieren soll, wie und wie sicher ein Multi-Applikations-Framework für Chipkarten aussehen kann. Die benötigten Teilkomponenten wurden nur insoweit betrachtet, als sie zur Erfüllung der primären Aufgabe relevant sind. Alle Teilkomponenten sind für sich betrachtet jedoch schon sehr komplex. Sicherlich gibt es dafür viele Ansätze und Experten. Wie genannt, habe ich versucht die Dienste so einfach und flexibel wie möglich zu halten. Wenn jemand eine bessere oder andere Implementierung besitzt, so kann er die prototypische Implementierung ganz einfach ersetzen. Dies wird durch das Service–Handler Konzept auf einfachste Weise ermöglicht.

7.1 Allgemeines

Als Programmiersprache wird Java eingesetzt [2]. Die Programmausgaben wurden in englisch gehalten. Dies liegt in der Möglichkeit begründet, das System auch internationalen Interessenten vorzuführen. Eine Anpassung der grundlegenden Programmausgaben an andere Sprachen ist durch eine bereits implementierte Klasse möglich. Diese kann so erweitert werden, dass die Ausgaben aus einer Datei gelesen werden. Alternativ ist die Einführung eines zusätzlichen Dienstes möglich, welcher entsprechend lokalisierte Ausgaben anbietet.

Der Quelltext der vorliegenden Arbeit wurde mit einer Quelltext-Versionsverwaltung gesichert. Dazu wurde das Concurrent Versions System, kurz CVS verwendet [6]. Als Server diente eine Windows-Implementation [7]. Als Client wurde entweder eine Kommandozeilenversion des Betriebssystems, sowie, unter Windows, WinCVS verwendet [8]. Bei der Erstellung der Packages wurde darauf geachtet, zum vorhandenen CVS–Quelltext des Campuskartenprojektes kompatibel zu bleiben. Dies wurde erreicht, indem alle Quelltexte dieser Arbeit unterhalb des Verzeichnisses `campuskarte/mafc` angesiedelt wurden.

7.2 Fehlerbehandlung

Ein grundlegendes Prinzip im Bereich der Fehlerbehandlung dieser prototypischen Implementierung ist eine maximale Transparenz. So viele Fehler wie möglich sollen bis zur Benutzerschnittstelle durchgeleitet werden. Zu diesem Zweck muss der Client die Klasse `campuskarte.mafc.lib.ErrorWebsite` auswerten können. In einem Produktionssystem sind unter Umständen nicht alle Fehlermeldungen für den Benutzer relevant. Zur genauen Fehlerdiagnose innerhalb eines Prototypen ist diese Vorgehensweise dennoch sehr sinnvoll.

Die in Abbildung 7.1 dargestellten, möglichen Fehlermeldungen bis auf `campuskarte.mafc.lib.exception.VOPException` dienen lediglich der Typisierung eines

[2] Der Entwurf bezieht sich hier auf weitergehende Verfeinerungen wie Objektinteraktionsdiagramme oder formale Beschreibungen.

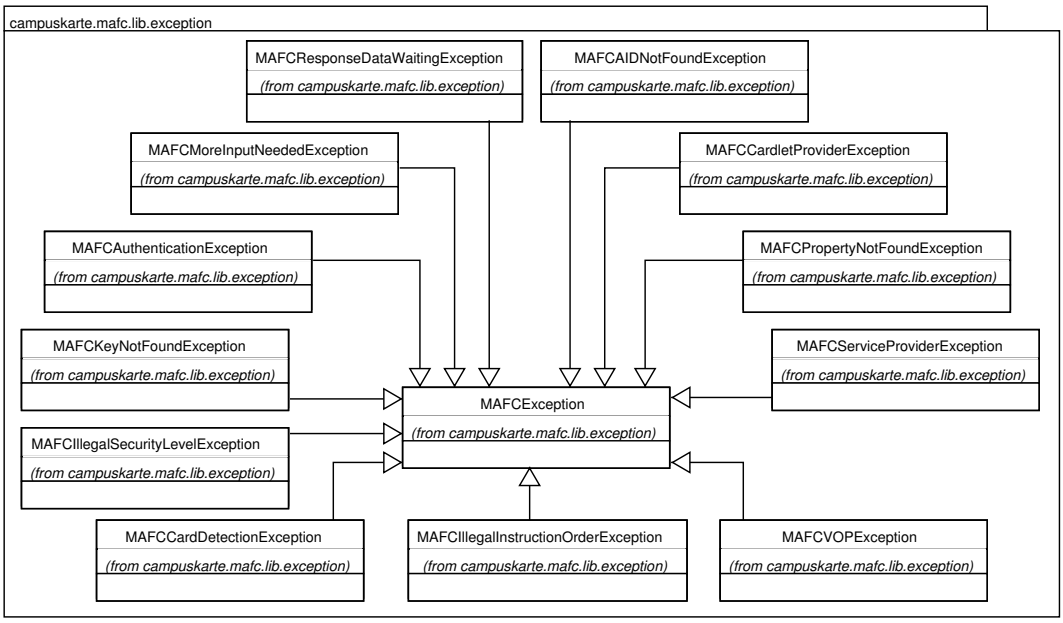

Abbildung 7.1: Klassendiagramm für verwendete Fehlertypen

Fehlers. `VOPException` behandelt Ausnahmefehler der Chipkarte und sichert den dabei erhaltenen Fehlercode. Eine genaue Beschreibung der Klassen liegt in der Java-Dokumentation zu dieser Arbeit vor.

7.3 Tests

Der Test eines Softwaresystem ist eine umfassende Aufgabe. Dies gilt erst recht für ein verteiltes System, wie es in dieser Arbeit entworfen wurde. Die Frage lautet nicht, ob getestet werden muss, sondern was und wie zur Zielerreichung getestet werden sollte. Auf höherer Ebene können Spezifikations- oder Model-Checking-Test durchgeführt werden, konkreter werden Funktions-, Modul- und Integrationstests [41].

Modultest für Java werden in [52] beschrieben. Darin wird eine enge Verknüpfung mit der eXtreme Programming Methode zur Softwareentwicklung dargestellt. Der Ansatz lautet: „Test first“. Zuerst sollte ein Unit-Test, welcher die Spezifikation des zu erstellenden Moduls überprüft und unter Umständen sogar erst definiert, geschrieben werden. Anschließend muss die Implementierung solange geändert werden, bis sie alle Testkriterien erfüllt. Dabei kommt auch ein wesentlicher Schwachpunkt dieser Methode zum Vorschein: Die Spezifikation muss lückenlos durch einen Unit-Test implementiert werden. Die Software, welche das Modul implementiert, könnte de facto per Zufallsgenerator erzeugt werden. Sobald die zufällig erzeugte Implementierung alle Tests besteht, ist das Modul fertig. Vom Prinzip wird das Hauptaugenmerk von der Implementierung zur Spezifikation verschoben, was eigentlich eine sehr schöne Sache ist. Leider besteht die Spezifikation für Module mit JUnit–Tests aus nichts anderem als Java-Programmen. Das bedeutet, diese Tests müssten wiederum auf ihre Korrektheit getestet werden. Es ist meines Erachtens sehr schwierig, eine allgemeine Spezifikation mit den Mitteln einer Programmiersprache wie Java zu erstellen.

Im Rahmen dieser Arbeit wurde mit Modultests begonnen. Dazu wurden die in den Schnittstellen definierten Funktionalitäten als JUnit-Tests implementiert. Für

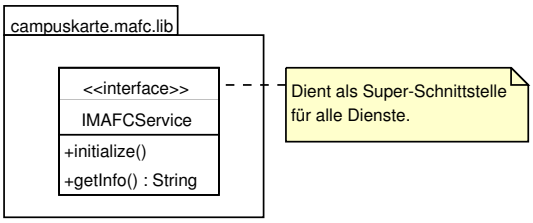

Abbildung 7.2: Super-Schnittstelle für Dienste

triviale oder leicht generierbare Tests, wie die (De)–Serialisierung von Objekten ließ sich dies auch sehr einfach durchführen. Sehr schwierig wurde es aber bei fast allen anderen Komponenten des Systems, welche nicht trivial waren. Dies lag darin begründet, dass sehr viele externe Komponenten wie Web-Server oder Chipkarten in Tests hätten einbezogen werden müssen, was mit JUnit wegen der Statik der Tests sehr schwierig ist. Ein Test der Schnittstellen kann auch sehr viel einfacher über das Java–Typsystem erfolgen. Letztendlich habe ich Modultests sehr schnell aufgegeben, einige vorhandene Tests befinden sich im Package `campuskarte.mafc.junit`.

Zum weiteren Testen habe ich mich auf Funktionstests beschränkt, welche das Framework umfassten. Dazu wurden verschiedene Kommandozeilenprogramme angelegt, die verschiedene Funktionalitäten der Dienste aufrufen. Diese befinden sich im Package `campuskarte.mafc.util`. Sie führen im Wesentlichen die Anwendungsfälle des Karten-Halters aus, namentlich Karteninhalt anzeigen (`campuskarte.mafc.util.listCardContent`), ein Cardlet installieren (`campuskarte.mafc.util.loadCardlet`) sowie ein Cardlet löschen (`campuskarte.mafc.util.deleteCardlet`). Dabei werden alle relevanten Dienste aufgerufen. Diese Tests lassen sich sehr schlecht mit JUnit abbilden, da externe Vorbedingungen gegeben sein müssen (z.B. muss ein Cardlet zum Löschen auf der Karte vorhanden sein). Gleichzeitig war dies ein erster Integrationstest der Dienste in das Multi-Applikations-Framework für Chipkarten.

7.4 Dienste

Das folgende Unterkapitel beschreibt die Implementierung aller im Entwurf enthaltenen Schnittstellen für die Dienste. Weitere, für die Implementierung notwendige, Dienste sind ebenfalls beschrieben.

Grundsätzlich sind alle Dienste Spezialisierungen der Schnittstelle `IMAFCService` (siehe Abbildung 7.2). Diese dient zum einen der Typisierung der Dienste und zum anderen der Bereitstellung von Initialisierungs- sowie Informationsfunktionalitäten. Die Funktionalität `initalize()` initialisiert den Dienst und `getInfo()` gibt eine textuelle Beschreibung des Dienstes zurück. Eine mögliche Erweiterung der Dienste könnte in die Richtung Java–Beans [47] gehen, wobei dann eine spezielle Konfiguration einfach graphisch „zusammengeklickt“ werden könnte.

Die Klasse `campuskarte.mafc.server.MAFCServiceHandler` implementiert den Service–Handler. Sie befindet sich unter dem Server-Package, da sie inhaltlich einen Dienste–Server bereitstellt. `MAFCServiceHandler` liest, wie alle anderen konfigurierbaren Dienste der prototypischen Implementierung, eine Konfigurationsdatei ein. Der Aufbau dieser Konfigurationsdateien ist in Kapitel 8 beschrieben. Wenn mittels der Funktionalität `getService()` ein spezieller Service angefordert wird, so muss der Schnittstellen-Name übergeben werden. Der `MAFCServiceHandler` ermittelt anhand der Konfigurationsdatei die den Dienst implementierende Klasse. Wurde diese

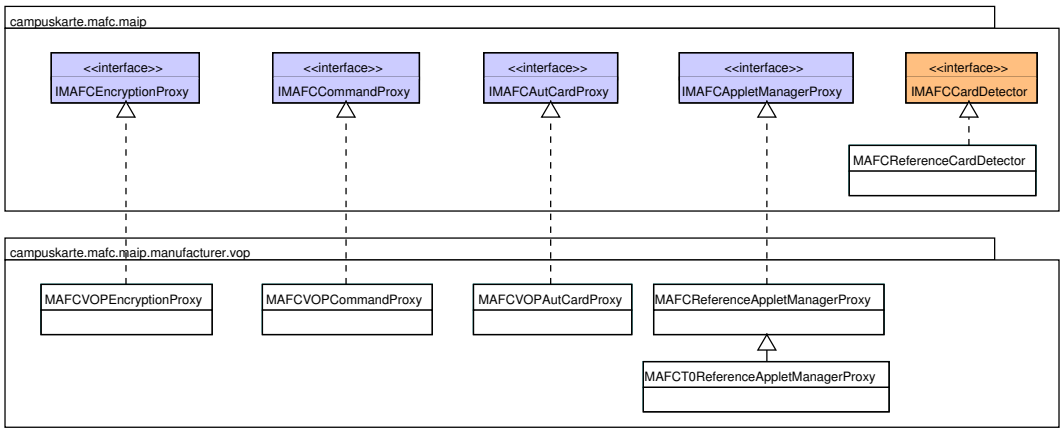

Abbildung 7.3: Klassendiagramm für VOP 2.0.1 kompatible Java-Karten

bereits einmal initialisiert, so wird sie einfach aus einer internen Tabelle geholt und zurückgegeben. Das bedeutet, dass für jede Instanz von `MAFCServiceHandler` jeder Dienst nur einmal erzeugt wird. Wurde die Klasse nicht gefunden, so wird versucht, sie mittels der Java-Runtime neu zu erzeugen. Gelingt dies, so wird überprüft ob sie dem Typ `IMAFCService` entspricht. Trifft auch dies zu, so wird die Funktionalität `initialize()` aufgerufen und die Klasse in einer internen Tabelle gespeichert. Anschließend wird die neu erzeugte Klasse zurückgegeben.

7.4.1 Multi-Applikations-Infrastruktur-Anbieter

Die zentrale Schnittstelle dieses Dienstes ist `IMAFCCardDetector`. Diese kann über den Service–Handler angefordert werden. Alle anderen Schnittstellen des Multi–Applikations–Infrastruktur–Anbieters werden kartenspezifisch durch `IMAFCCardDetector` initialisiert. Dies wird durch den Aufruf der `detectCard()` Funktionalität ausgelöst. Zur Erkennung des Kartentyps wird der ATR der eingelegten Karte verwendet. Sobald die Erkennung erfolgreich durchlaufen wurde, stehen die Schnittstellen `IMAFCEncryptionProxy`, `IMAFCCommandProxy`, `IMAFCAutCardProxy` sowie `IMAFCAppletManagerProxy` für den aktuellen Kartentyp durch den Service–Handler zur Verfügung.

Eine Referenzimplementation der Schnittstellen, welche der Visa Open Platform 2.0.1 Spezifikation entsprechen, befinden sich im Package `campuskarte.mafc.maip.manufacturer.vop` (siehe Abbildung 7.3). Eine gesonderte Betrachtung bedürfen der `MAFCReferenceAppletManagerProxy` sowie der `MAFCT0ReferenceAppletManagerProxy`. Der Erste implementiert die Schnittstelle `IMAFCAppletManagerProxy` für T=1 Karten, welche überwiegend auf ein Kommando mit einer APDU antworten. Der Zweite erweitert durch Vererbung die Funktionalität für T=0 Karten, welche auf ein Kommando zumeist mit zwei APDU's antworten. Diese benötigen ein spezielles Kommando zum Abholen der Antwortdaten.

Um die flexible Erweiterbarkeit für neue Kartentypen aufzuzeigen, werden in der prototypischen Implementierung drei verschiedene Java-Karten unterstützt. Sämtliche Karten werden auch mit dem vorhandenen Campuskartenframework unterstützt. Dies geschah dort jedoch durch sehr viel redundanten Quelltext. Anhand der folgenden Beschreibungen soll aufgezeigt werden, wie mit dem hier entwickelten Framework eine Erweiterung auf neue Karten–Typen auf einfachste Weise möglich ist.

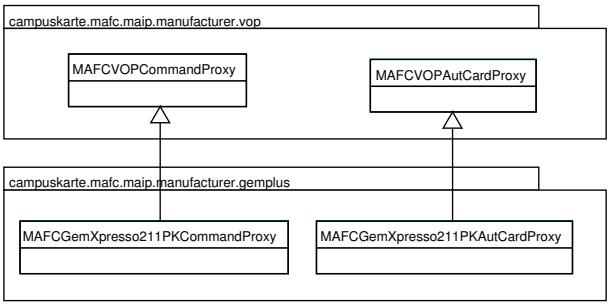

Abbildung 7.4: Klassendiagramm für Gemplus GemXpresso 211PK T=0 Java-Karte

Die Konfiguration kartenabhängiger Klassen erfolgt durch eine Konfigurationsdatei, welche im Kapitel Installation, Konfiguration und Demonstration beschrieben ist.

Gemplus GemXpresso 211PK T=0 Java–Karten

Die Gemplus GemXpresso 211PK T=0 Java–Karte besitzt eine proprietäre Schlüsselableitung sowie teilweise abgeänderte Kommandos [16]. Die Schlüsselableitung für diesen Kartentyp wurde in der Klasse `MAFCGemXpresso211PKAutCardProxy` implementiert, welche die entsprechenden Methoden der Superklasse (`MAFCVOPAutCardProxy`) überschreibt. Entgegen der VOP 2.0.1 Spezifikation benötigt die GemXpresso 211PK T=0 bei den meisten Kommandos keine LE-Bytes. Diese werden durch die Wrapper-Klasse `MAFCGemXPresso211PKCommandProxy` entsprechend abgeändert. Eine Darstellung der Klassen erfolgt in Abbildung 7.4.

Folgende Klassen werden durch die Kartenerkennung für eine GemXpresso 211 PK T=0 Java-Karte initialisiert:

- `IMAFCCommandProxy=`
  `campuskarte.mafc.maip.manufacturer.gemplus.`
  `MAFCGemXPresso211PKCommandProxy`
- `IMAFCAutCardProxy=`
  `campuskarte.mafc.maip.manufacturer.gemplus.`
  `MAFCGemXpresso211PKAutCardProxy`
- `IMAFCEncryptionProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.`
  `MAFCVOPEncryptionProxy`
- `IMAFCAppletManagerProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.`
  `MAFCTOReferenceAppletManagerProxy`

Gemplus GemXpresso Pro R3 T=0 Java–Karten

Die Gemplus GemXpresso Pro R3 T=0 Java–Karte ist vollständig VOP 2.0.1 konform. Lediglich zur Erkennung des freien Speichers ist eine angepasste Funktion notwendig, da diese nicht in VOP 2.0.1 spezifiziert ist. Eine entsprechende Funktionalität der Schnittstelle `IMAFCAppletManagerProxy` ist durch `getFreeMemory()` gegeben. In der Referenzimplementation (`MAFCReferenceAppletManagerProxy`) wirft

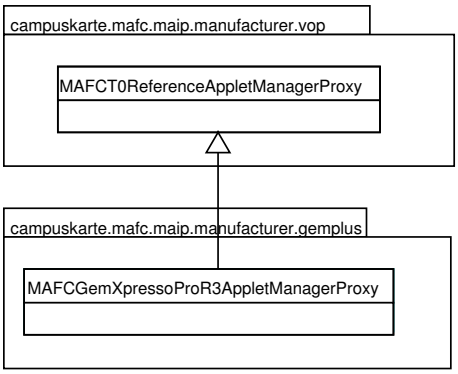

Abbildung 7.5: Klassendiagramm für Gemplus GemXpresso Pro R3 T=0 Java-Karte

diese standardmäßig einen `java.lang.UnsupportedOperationException` Ausnahmefehler.

Zur kartenspezifischen Anpassung wurde eine Spezialisierung der Klasse `MAFCT0-ReferenceAppletManagerProxy` vorgenommen, da die GemXpresso Pro R3 eine T=0 Java-Karte ist. Die Klasse `MAFCGemXpressoProR3AppletManagerProxy` stellt eine Methode zum Abfragen des freien Speichers der GemXpresso Pro R3 T=0 nach [17] zu Verfügung. Eine Darstellung der Klassen erfolgt in Abbildung 7.5.

Folgende Klassen werden durch die Kartenerkennung für eine GemXpresso Pro R3 T=0 Java-Karte initialisiert:

- `IMAFCCommandProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPCommandProxy`
- `IMAFCAutCardProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPAutCardProxy`
- `IMAFCEncryptionProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPEncryptionProxy`
- `IMAFCAppletManagerProxy=`
  `campuskarte.mafc.maip.manufacturer.gemplus.`
  `MAFCGemXpressoProR3AppletManagerProxy`

G&D Sm@rtCafe Expert 2.0 T=1 Java–Karten

Die G&D Sm@rtCafe Expert 2.0 T=1 Java–Karte entspricht ebenfalls vollständig der VOP 2.0.1 Spezifikation. Die Ermittelung des freien Kartenspeichers ist aber auch hier herstellerabhängig. Da die Sm@rtCafe eine T=1 Karte ist, wurde die Klasse `MAFCReferenceAppletManagerProxy` zu `MAFCSmartcafe2AppletManagerProxy` abgeleitet. Die Implementation der Methode `getFreeMemory()` erfolgte nach [18]. Eine Darstellung der Klassen erfolgt in Abbildung 7.6.

Folgende Klassen werden durch die Kartenerkennung für eine GemXpresso Pro R3 T=0 Java-Karte initialisiert:

- `IMAFCCommandProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPCommandProxy`

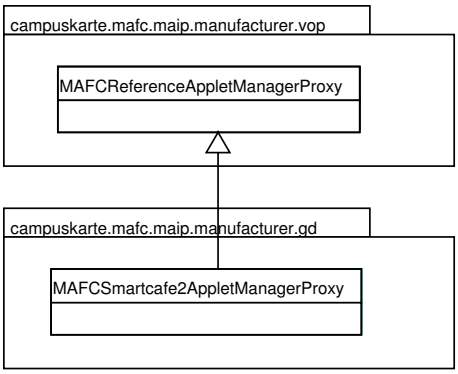

Abbildung 7.6: Klassendiagramm für G&D Sm@rtCafe Expert 2.0 T=1 Java Karten

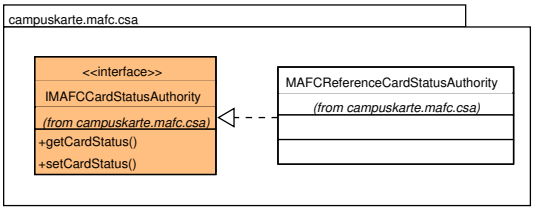

Abbildung 7.7: Klassendiagramm für das Karten–Status–Abfrage–System

- `IMAFCAutCardProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPAutCardProxy`
- `IMAFCEncryptionProxy=`
  `campuskarte.mafc.maip.manufacturer.vop.MAFCVOPEncryptionProxy`
- `IMAFCAppletManagerProxy=`
  `campuskarte.mafc.maip.manufacturer.gd.`
  `MAFCSmartcafe2AppletManagerProxy`

7.4.2 Karten–Status–Abfrage–System

Die Schnittstelle `IMAFCCardStatusAuthority` wird durch die Klasse `MAFCReferenceCardStatusAuthority` implementiert. Diese ist in Abbildung 7.7 dargestellt. Eine Abfrage des Status der eingelegten Karte erfolgt anhand der CUID der Chipkarte. Die Funktionalität `setStatus()` der Schnittstelle wird in der prototypischen Implementierung nicht unterstützt. Die Verwaltung des Kartenstatus erfolgt über eine Konfigurationsdatei, welche in Kapitel 8.2.2 beschrieben wird.

7.4.3 Cardlet–Anbieter

Die Schnittstelle des Cardlet–Anbieters wird durch die Klasse `MAFCReferenceCardletProvider` implementiert. Diese ist in Abbildung 7.8 dargestellt. Der einzige unterstützte Cardlettyp der prototypischen Implementierung ist ein JavaCard 2.1 kompatibles Cardlet–Format. Dieses wird durch die Klasse `JC21CardletCode` des Multi–Applikations–Infrastruktur–Anbieters bereitgestellt. Die Konfiguration der Meta–Informationen sowie der verfügbaren Cardlets wird in Kapitel 8.2.2 beschrieben.

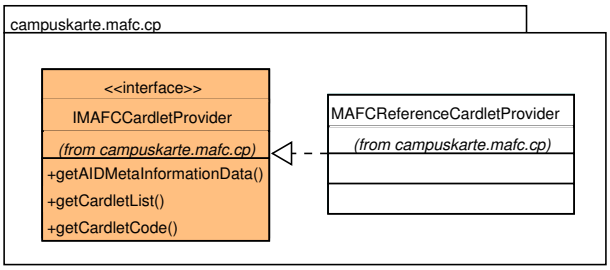

Abbildung 7.8: Klassendiagramm für den Cardlet–Anbieter

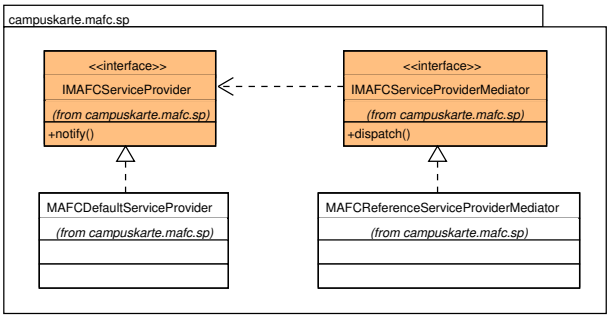

Abbildung 7.9: Klassendiagramm für den Anwendungs–Betreiber

7.4.4 Anwendungs–Betreiber

Der Anwendungsbetreiber besitzt zwei Schnittstellen. Die erstere, `IMAFCServiceProvider` dient der Benachrichtigung des Anwendungs–Betreibers über die Installation oder das Löschen von Cardlets. Diese wird durch die Klasse `MAFCDefaultServiceProvider` implementiert (siehe Abbildung 7.9). Der `MAFCDefaultServiceProvider` besitzt keine interne Funktionalität, was bedeutet, er „schluckt“ alle Eingaben wie `/dev/null`.

Die zweite Schnittstelle, `IMAFCServiceProviderMediator` entstammt, wie der Name bereits aussagt, dem Mediator-Pattern [51]. Dieses vermittelt eine Nachricht an den „passenden“ Anwendungsbetreiber. Die Schnittstelle wird implementiert durch die Klasse `MAFCReferenceServiceProviderMediator`. Zur flexibleren Konfiguration registrieren sich mögliche Observer jedoch nicht direkt beim `MAFCReferenceServiceProviderMediator`, sondern über eine Konfigurationsdatei, welche bei der Erzeugung der Klasse ausgelesen wird. Die Konfiguration wird in Kapitel 8.2.2 beschrieben.

7.4.5 Kartenschlüssel–Management

Die Schnittstelle `IMAFCKeyProvider` wird durch die Klasse `MAFCReferenceKeyProvider` implementiert. Diese ist in Abbildung 7.10 dargestellt. Die prototypische Implementierung unterstützt die Funktionalität `getKMC()` nicht. Diese stellt für einen bestimmten Kartentyp (ATR) und kartenspezifische Daten (CPLC) einen Hauptschlüssel (KMC) zur Verfügung (siehe Kapitel 6.6.3). Um dies zu ermöglichen, müssten die Daten chargenabhängig direkt nach der Lieferung erfasst werden. Dies wird durch die vorhandene Campuskarteninfrastruktur z.Z. nicht durchgeführt.

Unterstützt wird die Abfrage von kartenspezifischen Schlüsseln (KDC) anhand der CUID. Diese werden ebenfalls über eine Konfigurationsdatei gesetzt und in Ka-

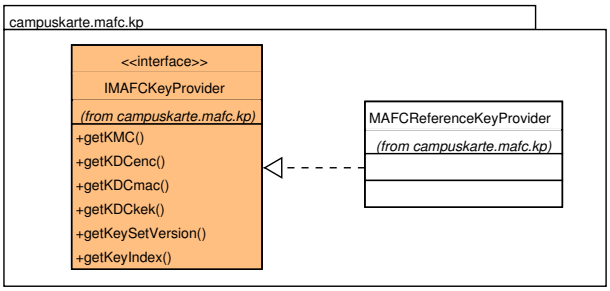

Abbildung 7.10: Klassendiagramm für das Kartenschlüssel–Management

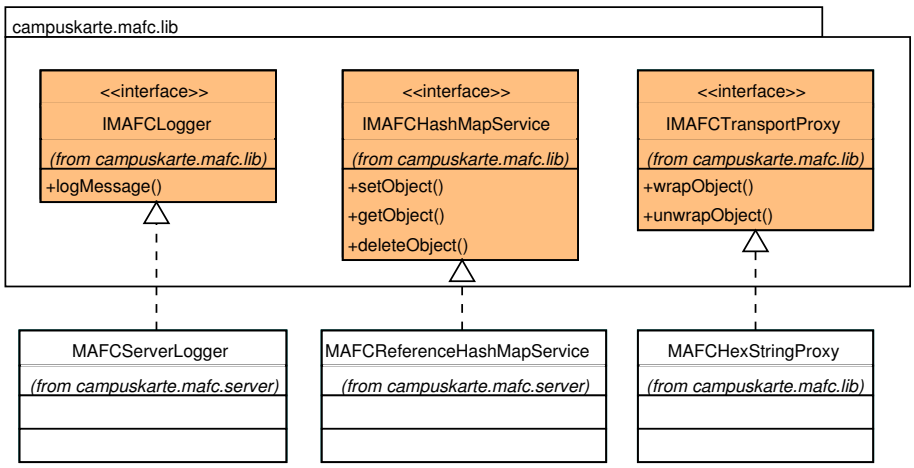

Abbildung 7.11: Schnittstellen und Implementationen für weitere Dienste

pitel 8.2.2 beschrieben.

7.4.6 Weitere Dienste

Zur Unterstützung der Systemfunktionalität wurden drei weitere Dienste implementiert. Diese sind in Abbildung 7.11 dargestellt.

Der erste Dienst besitzt die Schnittstelle `IMAFCLogger`. Er stellt eine Logger-Funktionalität zur Verfügung. Die Klasse `MAFCServerLogger` implementiert die Schnittstelle für den Server. Die Ausgabe erfolgt dabei auf `stdout`. Eine weitere Implementierung für den Client (`campuskarte.mafc.client.frame.MAFCLogWindowFrame`), stellt die Ausgaben in einem Fenster dar.

Der zweite Dienst besitzt die Schnittstelle `IMAFCHashMapService`. Sie wird implementiert durch die Klasse `MAFCReferenceHashMapService`. Er stellt eine, je Service–Handler Instanz, globale HashMap zur Datenspeicherung zur Verfügung.

Der dritte Dienst stellt eine Schnittstelle zur Objektserialisierung zur Verfügung (`IMAFCTransportProxy`). Er wird implementiert durch die Klasse `MAFCHexStringProxy`, welche Objekte auf das Java–Serialisierungsformat abbildet und daraus einen Hex–String erzeugt. Diese Schnittstelle kann z.B. auch durch eine Java-XML oder SOAP Objektserialisierung ersetzt werden.

Die genannten Dienste sind jederzeit über den Service–Handler verfügbar.

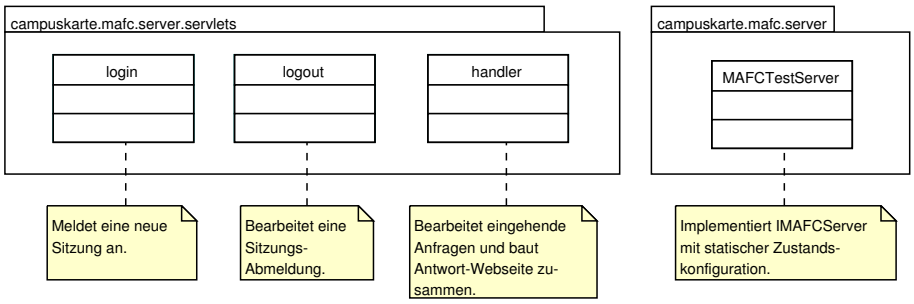

Abbildung 7.12: Klassendiagramm für den Server

7.5 Server

Der Server basiert auf drei Java–Servlets, welche als Schnittstelle zum Client dienen. Diese sind in Abbildung 7.12 dargestellt. Das Servlet `login` meldet eine neue Sitzung an. Durch den Aufruf werden eine Sitzungskennung sowie die URL's für die anderen Servlets zurückgegeben. Diese akzeptieren nur Anfragen mit einer gültigen Sitzungskennung. Durch den Aufruf von `logout` mit einer gültigen Sitzungskennung wird die Sitzung beendet. Das Servlet `handler` bearbeitet HTTP-POST Anfragen. Dabei werden der angeforderte Zustandsname sowie die dazugehörigen Parameter ausgepackt und an die Schnittstelle `IMAFCServer` übergeben. Wenn während der Bearbeitung ein Fehler auftritt, so wird dieser abgefangen und in einer `ErrorWebsite` gesichert. `ErrorWebsite` wird ebenso wie die Rückgabewerte von `IMAFCService` serialisiert und in eine Antwort–Webseite des Servlets gepackt. Diese wird an den Client übertragen.

Die eigentlich Arbeit des Server wird über die Schnittstelle `IMAFCServer` geleistet. Die Implementierung dieser Schnittstelle wird ebenfalls über den Service–Handler konfiguriert. In der prototypischen Implementierung ist dies `MAFCTestServer`, welche ebenfalls in Abbildung 7.12 dargestellt ist. Die Konfiguration eines zustands-basierten Servers kann auf sehr vielfältige Art und Weise geschehen. Eine mögliche Lösung ist der tabellen–basierte Ansatz. Es gibt dabei initiale Zustände, welche zu neuen (nicht–initialen) Zuständen führen können. Die Konfiguration dieses Zustandsgraphen kann entweder statisch oder dynamisch erfolgen. Bei der statischen Konfiguration wird der Zustandsgraph z.B. aus einer Konfigurationsdatei ausgelesen. Anhand der möglichen Pfade sind die jeweils erlaubten Zustände eindeutig definiert. Bei der dynamischen Konfiguration sind nur die initialen Zustandsobjekte festgelegt. Jedes Zustandsobjekt kennt seinen logischen Nachfolger und kann diesen zurückgeben. Dieser ist aber nicht statisch festgelegt, sondern wird dynamisch während der Ausführung entschieden.

Der `MAFCTestServer` besitzt eine dynamische Konfiguration, lediglich die initialen Zustandsobjekte sind fest einkodiert. Die dynamische Konfiguration ist erforderlich, weil die Zustandsobjekte nicht seriell hintereinander aufgerufen werden, sondern sehr oft ein Zustandsobjekt mehrmals hintereinander angefordert wird (siehe Abbildung 6.10).

Durch die Entscheidung, für jede Sitzung einen neuen Service–Handler zu erzeugen, konnte die Sitzungsverwaltung komplett aus dem restlichen Quelltext herausgehalten werden. Weiterhin wird durch diese Vorgehensweise eine hohe Datenkapselung mit geringer Fehlerwahrscheinlichkeit geschaffen.

Die in Abbildung 7.13 dargestellten Klassen implementieren Zustandsobjekte,

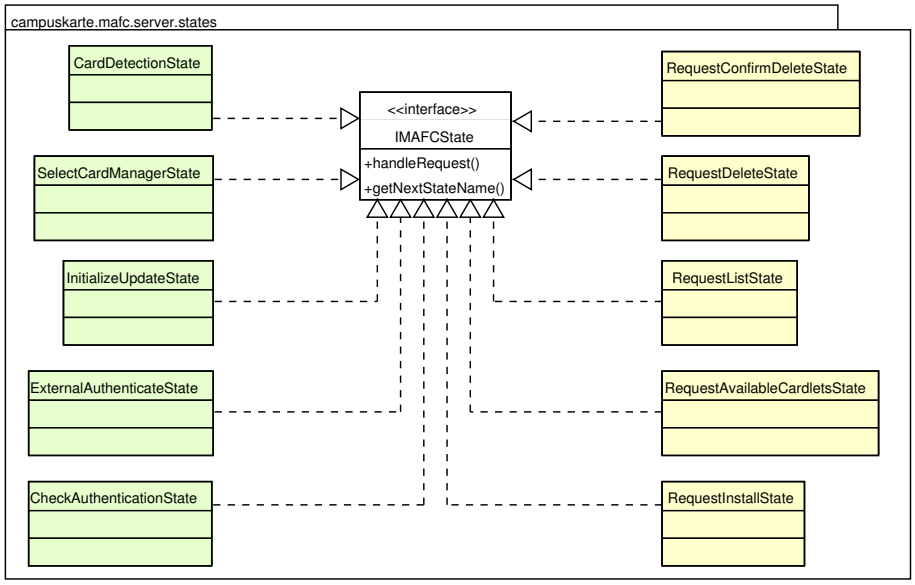

Abbildung 7.13: Klassendiagramm für die Server–Zustände

welche auf den in Kapitel 6.3 beschrieben Zustandsdiagrammen basieren.

Die Klassen `CardDetectionState`, `SelectCardManagerState`, `InitializeUpdateState`, `ExternalAuthenticateState` sowie `CheckAuthenticationState` implementieren den Ablauf des Zustandsdiagrammes Kartenauthentisierung des Servers (siehe Abbildung 6.9). Dazu greifen sie auf verschiedene Dienste zu.

Die Klasse `RequestListState` implementiert den Ablauf des Zustandsdiagrammes Karteninhalt anzeigen. Der Zustand besitzt mehrere Unterzustände, wobei diese über ein flaches Gedächtnis angesprochen werden. Der ausgelesene Karteninhalt wird mittels des Dienstes `IMAFCHashMapService` für die jeweilige Sitzung gesichert.

Die Klassen `RequestAvailableCardletsState` und `RequestInstallState` implementieren den Ablauf des Zustandsdiagrammes Cardlet installieren (siehe Abbildung 6.10). Die erste Klasse erzeugt eine Webseite, welche alle verfügbaren Cardlets auflistet, wobei bereits vorhandene unterdrückt werden. Dies geschieht mit Hilfe des im Dienst `IMAFCHashMapService` abgelegten Karteninhalts. Der zweite Zustand besitzt wiederum mehrere Unterzustände, welche verschiedene Kommandos zum Aufspielen des Cardlets generieren und auswerten. Nach erfolgreichem Aufspielen wird der Anwendungs–Betreiber benachrichtigt.

Die Klassen `RequestConfirmDeleteState` und `RequestDeleteState` implementieren den Ablauf des Zustandsdiagrammes Cardlet löschen (siehe Abbildung 6.10). Die erste Klasse erzeugt lediglich eine Webseite, in welcher der Benutzer gefragt wird, ob er das gewählte Cardlet wirklich löschen will. Die zweite Klasse erzeugt Kommandos zum Löschen der Application sowie des zugehörigen Loadfiles. Zuvor wird nochmals überprüft, ob das Cardlet wirklich gelöscht werden darf.

7.6 Client

Der Client basiert auf einem Java–Applet, welches im Web–Browser des Benutzers ausgeführt wird. Damit dieses Zugriff auf den Kartenleser bekommen kann, muss es digital signiert werden. Die wesentlichen Klassen des Applets sind in Abbildung 7.14

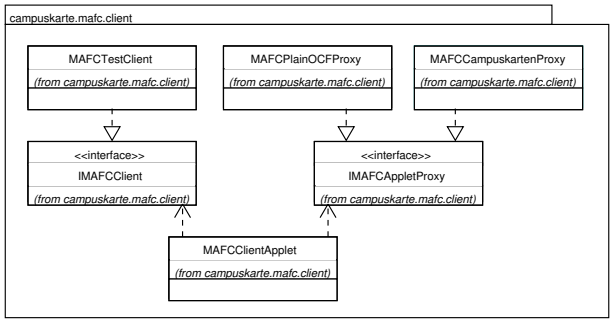

Abbildung 7.14: Klassendiagramm für den Client

dargestellt.

Die Klasse `MAFCClientApplet` ist die Hauptklasse des Applets. Diese verwendet zwei Schnittstellen, `IMAFCClient` sowie `IMAFCAppletProxy`, welche in Kapitel 6.4 beschrieben sind. Die Implementierung der Schnittstelle `IMAFCClient` wurde in `MAFCClientApplet` fest auf die Klasse `MAFCTestClient` gelegt. Dies erfolgte aufgrund eines möglichst einfachen und kompakten Aufbau des Clients. Möglich wäre eine Konfiguration der implementierenden Klassen auch über Applet–Parameter. Für die Implementierung der Schnittstelle `IMAFCAppletProxy` stehen zwei Klassen zur Verfügung, `MAFCPlainOCFProxy` sowie `MAFCCampuskartenProxy`.

Die Klasse `MAFCPlainOCFProxy` dient dem Zugriff auf die Chipkarte über den Opencard–`PassThruCardService`. Dieser ist für alle Chipkartentypen geeignet, muss jedoch durch den Benutzer in der Opencard Konfigurationsdatei `opencard.properties` konfiguriert werden.

Zur Verwendung der Campuskarten–Installation dient die Klasse `MAFCCampuskartenProxy`. Diese verwendet den Campuskarten–Proxy `CampuskarteAppletProxy` zum Zugriff auf die Chipkarte. Dieser Proxy ist aus technischen Gründen nur zur Verwendung mit Chipkarten, welche das Campuskarten–Cardlet enthalten, geeignet.

Das `MAFCClientApplet` selbst stellt im Wesentlichen eine `JEditorPane` zur Verfügung, welche zur Anzeige von Webseiten genutzt wird. Sämtliche Navigation erfolgt über Hyperlinks innerhalb der Webseite. Die Auswertung der Kommunikation erfolgt nach dem Entwurf in Kapitel 6.6.

Kapitel 8

Installation, Konfiguration und Demonstration

Im diesem Kapitel wird die Installation, Konfiguration und Demonstration der prototypischen Implementierung vorgestellt. Ein wesentliches Augenmerk wird auf die Konfiguration der implementierten Dienste gelegt, die Installation und Konfiguration des Web–Servers Tomcat wird nur am Rande betrachtet.

8.1 Kommandozeilenprogramme

Zum Testen der prototypischen Implementierung wurden drei Kommandozeilenprogramme geschrieben. Diese ermöglichen die lokale Rekonfiguration der Chipkarte und können z.B. zur Vorpersonalisierung erweitert werden. Ein weiteres Programm dient der kartenspezifischen Ableitung von KDC–Schlüsseln aus dem KMC–Schlüssel. Diese Funktionalität wird zur Konfiguration des Kartenschlüssel–Management benötigt. Alle hier genannten Programme befinden sich im Package `campuskarte.mafc.util`.

8.1.1 Programm `deriveKDC`

Das Programme `deriveKDC` leitet kartenspezifische Schlüssel (KDC) für unterstützte Java–Karten ab. Dazu wird der kartenspezifische Dienst `IMAFCAutCardProxy` verwendet. Als Eingabe wird der zur Karten–Charge gehörende KMC verwendet. `DeriveKDC` überprüft nicht die Gültigkeit von Schlüsseln. Ein beispielhafter Aufruf von `deriveKDC` ist in Abbildung 8.1 dargestellt.

```
java campuskarte.mafc.util.deriveKDC smartcafe2.isk
KDC key derivation for MAFC
Reading keyfile smartcafe2.isk...OK
KMC=404043434545464649494A4A4C4C4F4F4040434345454646
ATR=3BEA00FF8131FE455455422D434B010301007B
Card type=Giesecke & Devrient Sm@rtcafe Expert 2.0
AIDCardManager=A000000003000000
SELECT CARDMANAGER...OK
CUID=40900015232545093628
KDCenc=FDF8B5EF46700173F16D2545678594C7FDF8B5EF46700173
KDCmac=A78920583BDCDC0D0E85A292A289B376A78920583BDCDC0D
KDCkek=52DA13D523D6800864AE01BC1A9D8F7952DA13D523D68008
```

Abbildung 8.1: Beispielhafter Aufruf von `deriveKDC`

```
java campuskarte.mafc.util.listCardContent
List Card Content for MAFC
ATR=3BEA00FF8131FE455455422D434B010301007B
Card type=Giesecke & Devrient Sm@rtcafe Expert 2.0
AIDCardManager...A000000003000000
SELECT CARDMANAGER=OK
CUID=40900015232545093628
Card Status=OK
Mutual Authentication...OK
--- CARD CONTENT ---
A000000003000000..... CM OP_READY ( CARD_MANAGER_LOCK
CARD_MANAGER_TERMINATE PIN_CHANGE SECURITY_DOMAIN ) Meta-Inf: G&D Card-Manager
(Icon-Url: pics/gd.gif, Deleteable: false, Updateable: false, Loadfile: , Size:
0, Description: Administrates the chipcard.)
010203040502..... AP SELECTABLE () Meta-Inf: Business Card
(Icon-Url: pics/vcard.gif, Deleteable: true, Updateable: false, Loadfile:
010203040501, Size: 2048, Description: Sample business card application from
IBM.)
010203040501..... LF LOADED () Meta-Inf: Business Card Loadfile
(Icon-Url: pics/vcard.gif, Version: 100, Codesize: 1933, Signatur: )
A000000078584C50..... LF LOADED () Meta-Inf: n/a
A000000140544F444F53..... LF LOADED () Meta-Inf: n/a
Free eeprom space=10978 bytes.
```

Abbildung 8.2: Beispielhafter Aufruf von `listCardContent` (gekürzt)

8.1.2 Programm `listCardContent`

Das Programm `listCardContent` listet das Inhaltsverzeichnis einer Java–Karte auf. Abbildung 8.2 stellt eine gekürzte Ausgabe des Programmaufrufs dar. Der komplette Karteninhalt besteht aus weiteren Loadfiles, welche Systemfunktionalitäten bereitstellen.

Dieses, sowie die folgenden Programme, nutzen verschiedene Dienste des Multi–Applikations–Infrastruktur–Anbieters zur Erbringung der Funktionalität. Weiterhin werden das Karten–Status–Abfrage–System zum Abfragen des Kartenstatus sowie das Kartenschlüssel–Management zum Abfragen der Kartenschlüssel verwendet. Die Meta–Informationen zu den Loadfiles und Applications stammen vom Cardlet–Anbieter. Das Programm wird ohne Parameter aufgerufen.

8.1.3 Programm `loadCardlet`

Das Programm `loadCardlet` wird mit der AID der zu installierenden Application aufgerufen. Es wird versucht, den Programmcode über denCardlet–Provider zu erhalten. Dieser wird auf die Java–Karte geladen (Loadfile) und anschließend installiert (Application).

Wird das Programm ohne Parameter aufgerufen, so wird eine Liste der zur Installation verfügbaren Cardlets des Cardlet–Anbieters zurückgegeben. Ein Test auf die Kompatibilität mit bestimmten Chipkartentypen erfolgt im Kommandozeilenprogramm nicht.

8.1.4 Programm `deleteCardlet`

Das Programm `deleteCardlet` löscht ein bekanntes Cardlet von einer Java–Karte. Bekanntes Cardlet bedeutet, dass der Cardlet–Anbieter Meta–Informationen dazu bereitstellen kann. Nur mit Hilfe dieser Informationen kann festgestellt werden, ob das Cardlet gelöscht werden darf.

Name	Erlaubte Werte	Beschreibung
<String>	$x^* \mid x \in ASCII - Character$	Zeichenkette
<HexString>	$x^* \mid x \in \{0..9, A..F\}$	Binärdaten als Zeichenkette
<Boolean>	$x \mid x \in \{True, False\}$	Basistyp Boolean
<Byte>	$x \mid x \in \{0..255\}$	Basistyp Byte
<Int>	$x \mid x \in \{-2^{31}..2^{31}-1\}$	Basistyp Integer
<Url>	$x^* \mid x \in \{a..z, A..Z, 0..9, :, /, ?, \&, @, \_, .\}$	URL
<File>	$x^* \mid x \in \{a..z, A..Z, 0..9, ., /, \backslash, :, \_\}$	Dateiname mit Pfad
<Package>	$x^* \mid x \in \{a..z, A..Z, 0..9, ., \_\}$	Package–Name
OK	0	Konstante
LOCKED	1	Konstante
REVOKED	2	Konstante
<Status>	$x \mid x \in \{OK, LOCKED, REVOKED\}$	Kartenstatus
<ATR>	$< HexString >$	ATR einer Java–Karte
<CUID>	$< HexString >$	CUID einer Java–Karte
<AID>	$< HexString >$	AID einer *Application* oder eines *Loadfiles*
<Class>	$< Package >$	Java–Klassenname
<Interface>	$< Package >$	Java–Schnittstelle

Tabelle 8.1: Bezeichner und Typen für die Konfiguration der Dienste

Als Parameter wird die AID der zu löschenden Application angegeben. Das dazugehörige Loadfile wird aus den Meta–Informationen ermittelt und ebenfalls gelöscht.

8.2 Server Installation und Konfiguration

Dieser Abschnitt behandelt kurz die Installation und Konfiguration des Web–Servers Tomcat sowie ausführlicher die Konfiguration der prototypisch implementierten Dienste.

8.2.1 Web–Server

Der für die prototypische Implementierung des Multi–Applikations–Frameworks für Chipkarten verwendete Web–Server ist Tomcat. Dieser sollte auf einem Rechner innerhalb eines geschützten Netzabschnittes installiert werden. Für die prototypische Implementierung muss der Web–Server lediglich über den TCP/IP Port 443 (HTTPS) erreichbar sein. Alle anderen Ports sollten aus Sicherheitsgründen gesperrt werden.

Als Servlets müssen `login`, `logout` sowie `handler` registriert werden. Tomcat ist so zu konfigurieren, dass alle vom MAFC benötigten Konfigurationsdateien beim Start des Web–Servers im aktuellen Verzeichnis liegen. Weiterhin müssen alle benötigten Klassen, wie das Opencard Framework, im Suchpfad des Servers enthalten sein.

8.2.2 Dienste

Sämtliche Dienste werden durch das Hinzufügen des kompilierten Quelltextes des MAFC zum Web–Server–Klassenpfad eingebunden.

Die Konfiguration der verfügbaren Dienste erfolgt über die Konfigurationsdatei des Service–Handlers, `mafcservices.properties`. Diese enthält in jeweils einer Zeile den Namen der Schnittstelle und nach einem Gleichheitszeichen den kompletten Pfadnamen der implementierenden Klasse, zum Beispiel:

```
#
# Der vom Server verwendete Logger
#
campuskarte.mafc.lib.IMAFCLogger=campuskarte.mafc.server.MAFCServerLogger
```

Der Syntax ist im Folgendem dargestellt. Eine Beschreibung der verwendeten Bezeichner und Typen für sämtliche Dienstkonfigurationen ist in Tabelle 8.1 enthalten.

- *<Interface>=<Class>*

Multi–Applikations–Infrastruktur–Anbieter

Der Multi–Applikations–Infrastruktur–Anbieter fasst verschiedene Dienste zusammen. Diese werden kartenspezifisch mit Hilfe der Schnittstelle `IMAFCCardDetector` initialisiert. Die Konfiguration der verwendeten Klassen ist in der Datei `mafccardtypes.properties` festgelegt, zum Beispiel:[1]

```
#
# Gemplus GemXpresso 211PK T=0
#
3F6D000080318065B00501025E83009000.Name=Gemplus GemXpresso 211PK
3F6D000080318065B00501025E83009000.CPLCTag=9f7f
3F6D000080318065B00501025E83009000.CardManagerAID=A0000000018434D
3F6D000080318065B00501025E83009000.IMAFCCommandProxy=
campuskarte.mafc.maip.manufacturer.gemplus.MAFCGemXPresso211PKCommandProxy
3F6D000080318065B00501025E83009000.IMAFCAutCardProxy=
campuskarte.mafc.maip.manufacturer.gemplus.MAFCGemXpresso211PKAutCardProxy
3F6D000080318065B00501025E83009000.IMAFCEncryptionProxy=
campuskarte.mafc.maip.manufacturer.vop.MAFCVOPEncryptionProxy
3F6D000080318065B00501025E83009000.IMAFCAppletManagerProxy=
campuskarte.mafc.maip.manufacturer.vop.MAFCT0ReferenceAppletManagerProxy
```

Alle Einträge beginnen mit dem ATR der zu konfigurierenden Java–Karte. Zur Konfiguration der kartenspezifischen Dienste wird der Schnittstellenname hinter dem ATR mittels eines Punktes getrennt angegeben. Die implementierende Klasse folgt nach einem Gleichheitszeichen. Es müssen immer die folgenden Schnittstellen für jede unterstützte Java–Karte definiert werden:

- *<ATR>.IMAFCCommandProxy=<Class>*
- *<ATR>.IMAFCAutCardProxy=<Class>*
- *<ATR>.IMAFCEncryptionProxy=<Class>*

[1] Aus Layout–technischen Gründen erfolgt in den Darstellungen der Konfigurationsdateien teilweise ein Zeilenumbruch nach dem Gleichheitszeichen. Dieser darf in den verwendeten Konfigurationsdateien nicht enthalten sein.

- *<ATR>.IMAFCAppletManagerProxy=<Class>*

Zur korrekten Initialisierung der Karte werden zusätzlich der GetData–Kommando–Tag zum Abfragen der CPLC–Daten sowie die AID des Card–Managers benötigt. Weiterhin muss der Name der Java–Karte angegeben werden:

- *<ATR>.CPLCTag=<HexString>*
- *<ATR>.CardManagerAID=<HexString>*
- *<ATR>.Name=<String>*

Wenn ein ATR nicht komplett konfiguriert ist, oder nicht existiert, so wird die Karte vom MAFC nicht unterstützt.

Karten–Status–Abfrage–System

Das Karten–Status–Abfrage–System verwaltet in der prototypischen Version lediglich drei relevante Kartenzustände: `Ok`, `Gesperrt` und `Zurückgezogen`. Die Konfiguration erfolgt in der Datei `csa.properties`. Alle Einträge beginnen mit der CUID der jeweiligen Java–Karte gefolgt von einem Punkt sowie der Kennung STATUS und dem Status–Code, zum Beispiel:

```
40900015232545093628.STATUS=0
```

Der Syntax lautet:

- *<CUID>.STATUS=<Status>*

Cardlet–Anbieter

Der Cardlet–Anbieter wird über die Datei `cardletprovider.properties` konfiguriert. In einer Konfigurationsdatei können beliebig viele Applications und Loadfiles sowie eine Liste der verfügbaren Cardlets definiert werden. Beispielhaft sieht dies so aus:

```
#
# Business-Cardlet V1 Applikation
#
010203040502.TYPE=64
010203040502.NAME=Business Card
010203040502.DESCRIPTION=Sample business card application from IBM.
010203040502.ICON=pics/vcard.gif
010203040502.UPDATE=false
010203040502.DELETE=true
010203040502.LOADFILE=010203040501
010203040502.SIZE=2048
#
# Business-Cardlet V1 Loadfile
#
010203040501.TYPE=32
010203040501.NAME=Business Card Loadfile
010203040501.ICON=pics/vcard.gif
010203040501.CODESIZE=1933
010203040501.SIGNATURE=
```

```
010203040501.VERSION=100
#
# Verfügbare Cardlets (mit Komma trennen)
#
CARDLETS=cardlets/business/business.properties,cardlets/campuskarte/campuskarte.properties
```

Der Abschnitt für Applications setzt sich folgendermaßen zusammen: Alle Einträge beginnen mit der AID der Application. Auf diese folgen nach einem Punkt verschiedene Bezeichner. TYPE gibt den Typ der Applikation an, gültige Werte nach VOP 2.0.1 sind hier 64 (Application) sowie 128 (Card–Manager). Name gibt den Namen der Application an, während DESCRIPTION eine Beschreibung enthält. Mittels ICON wird eine URL zu einem Symbol definiert, welches die Application repräsentiert. UPDATE und DELETE bestimmen, ob die Application aktualisiert oder gelöscht werden darf. LOADFILE gibt die AID des Loadfiles zur Application an, sofern diese bekannt ist. Der Bezeichner SIZE gibt die Größe der installierten Application an. Sollte sich diese im ROM befinden, so ist der Wert 0 zu verwenden, -1 gibt eine unbekannte Größe an.

Der Syntax für Applications lautet:

- *<AID>.TYPE=<Byte>*
- *<AID>.NAME=<String>*
- *<AID>.ICON=<Url>*
- *<AID>.UPDATE=<Boolean>*
- *<AID>.DELETE=<Boolean>*
- *<AID>.LOADFILE=<HexString>*
- *<AID>.SIZE=<Int>*

Der Abschnitt für Loadfiles setzt sich syntaktisch wie der vorhergehende Abschnitt zusammen. Lediglich die Bezeichner sind teilweise verschieden. TYPE gibt den Typ an, ein gültiger Werte für Loadfiles ist nach VOP 2.0.1 hier 32. CODESIZE gibt die Größe des Loadfile an. SIGNATURE kann die digitale Signatur des Loadfiles enthalten, während VERSION die Versionsnummer angibt.

Der vollständige Syntax für Loadfiles lautet:

- *<AID>.TYPE=<Byte>*
- *<AID>.NAME=<String>*
- *<AID>.ICON=<Url>*
- *<AID>.CODESIZE=<Int>*
- *<AID>.SIGNATURE=<HexString>*
- *<AID>.VERSION=<Int>*

Der Abschnitts für verfügbare Cardlets besteht lediglich aus dem Bezeichner CARDLET, auf welchem, mittels Kommata getrennt, eine Liste der Konfigurationsdateien einzelner Cardlets folgen.

Der Syntax für die verfügbaren Cardlets lautet:

- *CARDLET=<File>, <File>, ...*

Die Konfigurationsdatei für ein verfügbares Cardlet enthält alle nach VOP 2.0.1 benötigten Informationen zum Laden und Installieren eines Cardlets. Beispielhaft sieht diese folgendermaßen aus:

```
#
# MAFC Konfigurationsdatei für das Business Cardlet
#
# Name des CAP-Files relativ zur Konfigurationsdatei
CAPFile = business.cap
# Pfad im CAP-File
CAPFilePackage = samples.business
# AID des Loadfiles
AIDLoadFile = 010203040501
# AID der installierten Applikation
AIDApplication = 010203040502
# AID der Security Domain
AIDSecurityDomain =
# DAP des Loadfiles
LoadFileDAP =
# Parameter für das Loadfile
LoadFileParameter =
# Load Token
LoadToken =
# Applikationsprivilegien (siehe VOP 2.0.1, Abschnitt 9-2)
ApplicationPrivileges = 0
# Installationsparameter
ApplicationInstallParameter =
# Das Install Token
InstallToken =
# Bekannte Karten, auf welchen diese Applikation verwendet werden darf.
# (Leer = alle)
RestrictedATRs =
```

Die Bedeutung der einzelnen Bezeichner entspricht den Parametern der Chipkarten–Kommandos, welche in [19, Abschn. 9-11 ff.] beschrieben werden. Eine Ausnahme bilden lediglich CAPFile sowie RestrictedATRs. Der erste Bezeichner gibt den Pfad und Dateinamen der zugehörigen CAP–Datei an. Durch RestrictedATRs kann eine mittels Kommata getrennte Liste von ATR's angegeben werden, zu welchen dieses Cardlet kompatibel ist.

Der vollständige Syntax für eine Cardlet–Konfigurationsdatei lautet:

- *CAPFile=<File>*
- *CAPFilePackage=<Package>*
- *AIDLoadFile=<HexString>*
- *AIDApplication=<HexString>*
- *AIDSecurityDomain=<HexString>*
- *LoadFileDAP=<HexString>*
- *LoadFileParameter=<HexString>*

- *LoadToken=<HexString>*
- *ApplicationPrivileges=<Byte>*
- *InstallToken=<HexString>*
- *RestrictedATRs=<HexString>, <HexString>, ...*

Anwendungs–Betreiber

Ein Anwendungs–Betreiber kann sich bei der Schnittstelle `IMAFCServiceProvider-Mediator` registrieren lassen, um über die Installation oder das Löschen eines zu ihm gehörigen Cardlets informiert zu werden. Die Konfiguration der prototypischen Implementierung, `MAFCReferenceServiceProviderMediator` erfolgt in der Datei `serviceprovider.properties`. Diese sieht beispielhaft wie folgt aus:

```
# IBM Business Cardlet
010203040502 = campuskarte.mafc.sp.MAFCDefaultServiceProvider
# Alle nicht zugeordneten Anwendungen werden an diese Klasse geleitet
default = campuskarte.mafc.sp.MAFCDefaultServiceProvider
```

Für eine bestimmte AID als Bezeichner kann jeweils eine Implementierung der Schnittstelle `IMAFCServiceProvider` angegeben werden. Der Bezeichner default gibt die Implementierung der `IMAFCServiceProvider`–Schnittstelle an, welche verwendet wird, wenn für eine AID kein Eintrag gefunden wurde.

Der Syntax lautet:

- *<AID>=<Class>*
- *default=<Class>*

Kartenschlüssel–Management

Die prototypische Implementierung des Kartenschlüssel–Managements, `MAFCReferenceKeyProvider` kann über die Datei `keydata.properties` konfiguriert werden. In dieser Datei werden die kartenabhängigen Card–Manager Schlüssel (KDC) gespeichert. Für eine Java–Karte sieht dies zum Beispiel so aus:

```
40900015232545093628.KDCENC=FDF8B5EF46700173F16D2545678594C7FDF8B5EF46700173
40900015232545093628.KDCMAC=A78920583BDCDC0D0E85A292A289B376A78920583BDCDC0D
40900015232545093628.KDCKEK=52DA13D523D6800864AE01BC1A9D8F7952DA13D523D68008
40900015232545093628.KEYSETVERSION=0
40900015232545093628.KEYINDEX=0
```

Alle Einträge beginnen mit der CUID der zu konfigurierenden Java–Karte. Zur Konfiguration der kartenspezifischen Schlüssel werden verschiedene Bezeichner durch einen Punkt getrennt angegeben. KDCENC, KDCMAC sowie KDCKEK geben die durch das Kommandozeilen–Programm `deriveKDC` abgeleiteten Schlüssel an. KEYSETVERSION gibt an, welche Schlüsselsetversion auf der Karte diesen Schlüsseln entspricht. Standardmäßig ist dies 0 (für das zuletzt aufgespielten Schlüsselset). Falls die Schlüssel in einer anderen Reihenfolge geladen wurden, so sieht VOP 2.0.1 einen Schlüsselindex vor, welcher den ersten Schlüssel angibt. Dies kann über den Bezeichner KEYINDEX konfiguriert werden. Der Standardwert beträgt auch hier 0.

Der Syntax lautet:

- *<CUID>.KDCENC=<HexString>*

Abbildung 8.3: Startbildschirm des Client–Applets

- *<CUID>.KDCMAC=<HexString>*
- *<CUID>.KDCKEK=<HexString>*
- *<CUID>.KEYSETVERSION=<Byte>*
- *<CUID>.KEYINDEX=<Byte>*

8.3 Client Installation und Konfiguration

Zur Installation des Client–PC genügt eine Campuskarten–Installation, welche Java ab Version 1.4 umfassen muss. Wie bereits im Kapitel Prototypische Implementierung beschrieben, existieren zwei Varianten zur Anbindung des Client–Applets an einen Kartenleser. Für die Verwendung mit dem Campuskarte–Setup ist die Schnittstelle `IMAFCAppletProxy` durch die Klasse `MAFCCampuskartenProxy` zu implementieren. Diese muss im Quelltext der Klasse `MAFCClientApplet` explizit angeben werden.

Das erstellte Applet ist mit weiteren benötigten Dateien (z.B. Opencard–Framework, Kartenleser–Treibern) zum Aufruf von einer Webseite des Web–Servers bereitzustellen.

8.4 Demonstration

Im Folgenden wird das prototypisch implementierte System anhand der Benutzerschnittstelle für den Karten–Halter demonstriert. Dazu werden die Anwendungsfälle des Karten–Halters betrachtet und die wesentlichen Bildschirmausgaben dargestellt und erläutert.

8.4.1 Client–Applet starten

Das Client–Applet wird durch den Aufruf der entsprechenden Web–Seite des Web–Servers gestartet. Sollte mehr als ein Kartenleser angeschlossen sein, so wird ein Dialog zur Auswahl angezeigt.

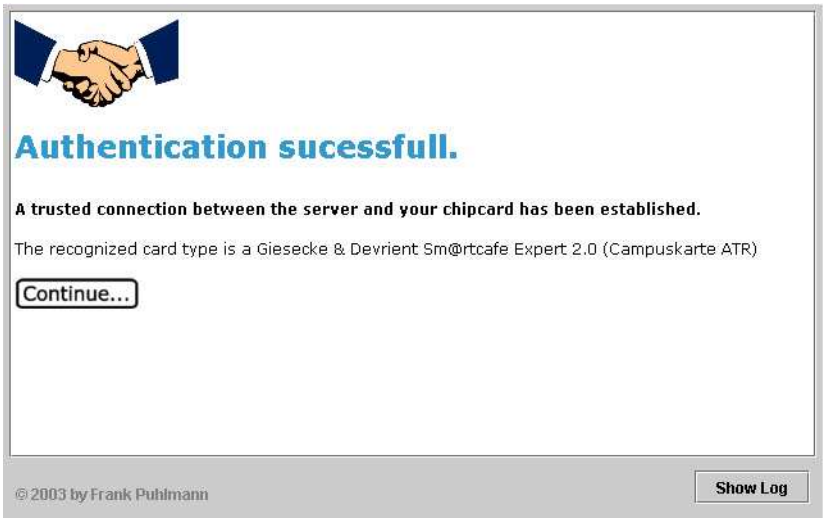

Abbildung 8.4: Erfolgreiche Authentisierung der Java–Karte

Der Startbildschirm des Client–Applets ist in Abbildung 8.3 dargestellt. Der Benutzer[2] wird zur Eingabe einer Chipkarte in den Karten–Leser aufgefordert. Mittels der „Show Log“ Schaltfläche kann ein Protokollfenster der übertragenen Daten zwischen Server und Client angezeigt werden. Dieses wird während der Übertragung in Echtzeit aktualisiert und enthält die verschlüsselten Kommandos für die Chipkarte, die Antworten sowie Benutzer–Aktivitäten.

Eine Entnahme der Chipkarte kann jederzeit erfolgen. Da jedoch die Konsistenz des Opencard Frameworks nicht immer gewahrt bleibt, wird empfohlen die Chipkarte nicht während der Anzeige von „Please wait...“ zu entnehmen. In diesem Moment finden Zugriffe auf die Chipkarte statt. Der Server bleibt in jedem Fall konsistent, gegebenenfalls muss aber der Web–Browser neu gestartet werden.

8.4.2 Kartenauthentisierung

Nach dem Einlegen einer Chipkarte wird automatisch der Anwendungsfall Kartenauthentisierung durchgeführt. Ist dieser erfolgreich, so wird eine Meldung, ähnlich der in Abbildung 8.4 dargestellten ausgegeben. Der Benutzer wird darauf hingewiesen, dass eine gesicherte Verbindung zwischen dem Server und der Chipkarte hergestellt wurde. Ferner wird der Typ der erkannten Chipkarte ausgegeben.

Im Fehlerfall wird eine entsprechende Meldung ausgegeben. Dies kann z.B. bei einem unbekannten Kartentyp, einer nicht registrierten oder gesperrten Chipkarte der Fall sein.

In jedem Fall muss der Benutzer auf die Schaltfläche „Continue...“ klicken, um fortzufahren.

8.4.3 Karteninhalt anzeigen

Die Visualisierung der Ergebnisse des Anwendungsfalls Karteninhalt anzeigen dient als zentraler Ausgangspunkt für alle weiteren Anwendungsfälle (siehe Abbildung 8.5). Im Gegensatz zum Kommandozeilenprogramm `listCardContent` werden hier

[2]Die Bezeichnung Benutzer wird im Folgenden als Vertretung des Karten–Halters verwendet.

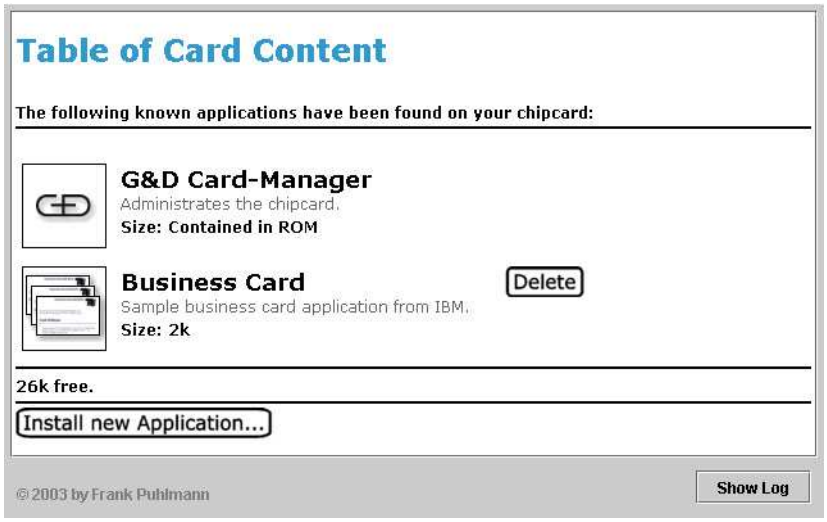

Abbildung 8.5: Anzeige des Karten–Inhaltes

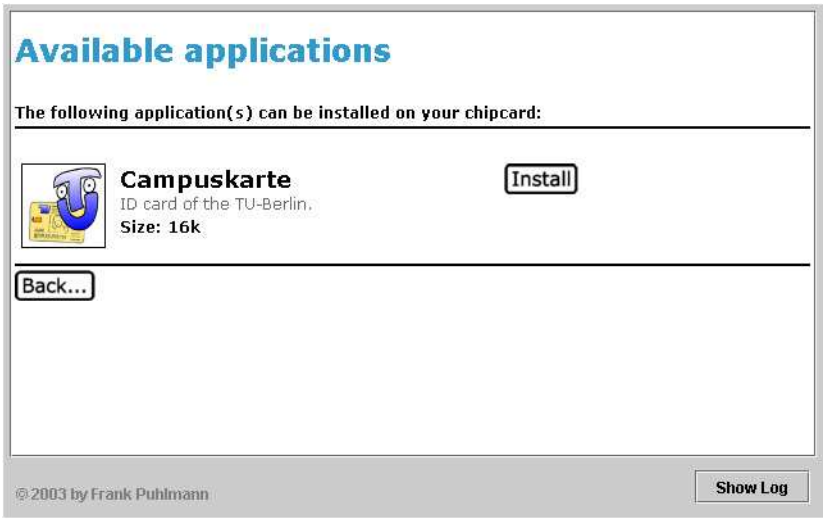

Abbildung 8.6: Verfügbare Cardlets zum Aufspielen auf die Java–Karte

nur die für den Benutzer relevanten Cardlets angezeigt. Diese werden mit verschiedenen Meta–Informationen, wie einem Icon, einem Namen, einer Beschreibung und der Größe, aufbereitet dargestellt.

Der Benutzer kann die beiden Anwendungsfälle Cardlet installieren sowie Cardlet löschen von dieser Darstellung aufrufen.

Die Schaltfläche Delete erscheint hinter jedem Cardlet–Eintrag, welcher nicht löschgeschützt ist. So darf z.B. der Card–Manager in Abbildung 8.5 aus Sicherheitsgründen nicht gelöscht werden. Technisch ist dies durchaus möglich, denn obwohl der Card–Manager im ROM installiert ist, kann er doch logisch gelöscht werden.

Die Schaltfläche „Install new Application...“ ruft den Anwendungsfall Cardlet installieren auf.

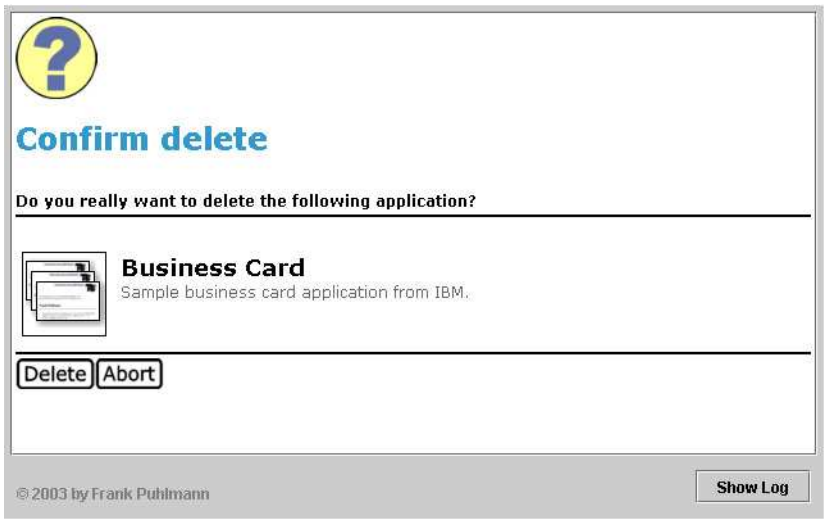

Abbildung 8.7: Cardlet löschen

8.4.4 Cardlet installieren

Der Anwendungsfall Cardlet installieren enthält den Anwendungsfall Verfügbare Cardlets anzeigen, dessen Ergebnis in Abbildung 8.6 dargestellt ist.

Dem Benutzer werden alle für seine Chipkarte geeigneten Cardlets angezeigt, welche er noch nicht installiert hat. Mit der Schaltfläche „Install“ kann das Cardlet installiert werden, mittels „Back...“ wird die Anzeige des Karteninhalts aufgerufen.

8.4.5 Cardlet löschen

In Abbildung 8.7 wird die Visualisierung eines Zwischenschrittes des Anwendungsfalls Cardlet löschen dargestellt. Der Benutzer wird gefragt, ob er das Cardlet wirklich löschen will. Die Schaltfläche „Delete“ ruft den endgültigen Löschvorgang auf, „Abort“ bricht den Vorgang ab.

Kapitel 9

Zusammenfassung

9.1 Ergebnisse

Diese Arbeit befasst sich mit der Rekonfiguration von Chipkarten nach der Ausgabe. Eine umfassende Einführung von Multi–Applikations–Chipkarten würde der Anwender zuerst in seiner Brieftasche bemerken — sie ist wesentlich dünner. Dies liegt jedoch nicht an den hohen Kosten einer solchen Chipkarte, sondern daran dass es nur noch eine einzige für mehrere Anwendungszwecke gibt. Diese Chipkarte kann vom Benutzer ganz nach seinen Wünschen konfiguriert werden, so wie er bisher entscheidet, welche Karten er in seine Brieftasche steckt. Für die Karten– und Anwendungs–Betreiber ergeben sich neue Möglichkeiten der schnellen und kostengünstigen Bereitstellung von neuen Anwendungen oder Erweiterungen. Es müssen keine neuen Chipkarten für neue Anwendungen hergestellt und ausgegeben werden; der ganze Aufwand beschränkt sich auf die Bereitstellung einer „virtuellen Karte“, in Form eines Programmes für eine Multi–Applikations–Chipkarte.

In dieser Arbeit wurde die technische und praktische „Machbarkeit“ eines Multi–Applikations–Frameworks für Chipkarten anhand eines Prototypen aufgezeigt. Dieser kann nahtlos in das bestehende Campuskartenframework der TU–Berlin integriert werden. Er demonstriert die performante und einfache Rekonfiguration einer Chipkarte durch den Karten–Halter. Die Voraussetzungen beschränken sich dabei auf einen Campuskarten kompatiblen Rechner, an welchem keine weiteren Änderungen oder Installationen nötig sind.

Der Prototyp basiert auf einem generischen, flexibel konfigurier– und erweiterbarem Framework, welches in dieser Arbeit ausführlich analysiert und entworfen wurde. Es bildet eine sehr gute Ausgangsbasis für die zukünftige Weiterentwicklung der Campuskarte. Es wurde eine Neuentwicklung zentraler Komponenten des existierenden Campuskartenframeworks durchgeführt, wodurch diese flexibler verwendet und erweitert werden können. Das neu entwickelte Framework ist z.B. nicht an eine Client–Server Architektur gebunden; neue Chipkartentypen können sehr einfach hinzugefügt werden.

Viele neue Komponenten und Funktionalitäten, welche in dieser Arbeit gefunden und analysiert wurden, ermöglichen eine Verwendung des Frameworks für Multi–Applikations–Chipkarten. Dies ist zum ersten der Multi–Applikations–Infrastruktur–Anbieter, welcher Kommandos für Chipkarten generiert und auswertet. Im Gegensatz zum existierenden Framework der Campuskarte ist dieser, durch die Trennung von Befehlsgenerierung und –auswertung, auch für verteilte Architekturen verwendbar. Diese wiederum sind eine Grundlage für die Rekonfiguration von Chipkarten

über offene Netze. Weiterhin wird eine klare Trennung zwischen dem Status einer Chipkarte und dem Status einer Anwendung auf der Chipkarte getroffen. Beide Varianten umfassen unterschiedliche Lebenszyklen. Im bisherigen Campuskartenprojekt wurde die Chipkarte sehr oft mit dem auf ihr installierten Programm, der Campuskartenanwendung, gleichgesetzt. Ein Karten–Status–Abfrage–System, welches den Lebenszyklus der Chipkarte als Container für weitere Programme verwaltet, ist unabdingbar für eine Multi–Applikations–Chipkarte. Die Bereitstellung kartenspezifischer Programme zum Aufspielen auf die Chipkarte sowie von Meta–Informationen zu diesen, wie z.B. Name, Größe und Schreibschutz, ist ebenso notwendig. Kartenspezifische Programme werden benötigt, da sich Java–Karten teilweise in kleinen Details unterscheiden und unterschiedliche Versionen eines Programmes benötigen. Auch die Größe des benötigten Speichers verhindert die Installation eines Programmes auf allen Chipkarten. Die Meta–Informationen werden benötigt, da heute erhältliche Chipkarten nur eine Bytefolge als Kennung eines Programmes speichern. Weiter gehende Informationen müssen aus Hintergrundsystemen gewonnen werden. Ein weiterer, sehr wichtiger Dienst ist ein Kartenschlüssel–Management. Zur Rekonfiguration einer Chipkarte, mit dem existierenden Framework, muss der zu verwendende Hauptschlüssel stets explizit angegeben werden. Der Benutzer muss die Schlüssel von Hand verwalten, kartenspezifische Schlüssel werden nicht unterstützt. Das in dieser Arbeit entwickelte Kartenschlüssel–Management stellt kartenspezifische Schlüssel automatisch zur Verfügung.

Diese Arbeit umfasst aber mehr als einen Prototypen zur Demonstration der Machbarkeit und ein flexibel erweiterbares Framework, indem ein großer Bogen über die Anforderungs– und Systemanalyse, den Entwurf und die prototypische Implementierung eines Multi–Applikations–Frameworks für Chipkarten gespannt wurde. Durch diesen weiten Blickwinkel konnten zusätzliche Problemstellungen, Ideen und Anregungen zum Thema gewonnen werden.

Ersichtlich wurden, bereits in der Protokollanalyse der Kommunikation zwischen Chipkarte und Rekonfigurations–Anwendung, Sicherheitsprobleme und Schwachstellen mit heute erhältlichen Chipkarten. Diese führen mit der Anwendung zwar eine gegenseitige Authentisierung durch, vergeben die dadurch gewonnene Sicherheit jedoch sofort wieder, indem die Kommunikation nur in Richtung Anwendung $\Rightarrow$ Chipkarte verschlüsselt wird. Aus Kompatibilitätsgründen zum ISO7816–Standard für Chipkarten ist zudem der Header, und damit der Typ eines Kommandos für die Chipkarte, unverschlüsselt zu übertragen. Alle Rückgabewerte erfolgen im unsignierten Klartext. Zur Lösung dieser Sicherheitsprobleme gibt es eine erweiterte Fassung des Visa Open Platform Standards [20], in welchem neue Kommunikationsprotokolle beschrieben werden. Momentan erhältliche Chipkarten unterstützen diesen neuen Standard bislang nicht.

Auch die gesellschaftlichen Auswirkungen der Einführung einer Multi–Applikations–Chipkarten müssen betrachtet werden. Zusätzlich zu den vorhandenen Vor– und auch Nachteilen von Chipkarten, wie die Möglichkeiten der Überwachung, beschleunigen und konzentrieren Multi–Applikations–Chipkarten alle positiven und negativen Effekte. Es werden noch mehr Arbeitsplätze in noch kürzerer Zeit durch Computer ersetzt werden können. Letztendlich kann dadurch aber auch eine Konzentration auf wesentliche, wertschöpfende Arbeiten erfolgen. Vor allem die so genannte „Verwaltung“ kann wesentlich reduziert werden. Dies ist gerade im öffentlichen Sektor, welcher unter hohen Kosten und geringer Produktivität leidet, eine sehr gute Möglichkeit, sich den Kernaufgaben zu widmen.

Zu den erkannten Problemen kommen aber auch viele neue Ideen und Anregungen, um Multi–Applikations–Chipkarten unter Wahrung der Schutzziele eines jeden Beteiligten zu verwenden. So dient die Aufteilung in die Rollen Anwendungs–Betreiber, Cardlet–Anbieter und Multi–Applikations–Infrastruktur–Betreiber dem maximalen Schutz jeder Rolle. Prinzipiell könnten verschiedene Rollen auch zusammengefasst werden. Damit aber keine Kompromittierung der Infrastruktur durch den Anwendungs–Betreiber stattfinden kann, darf dieser Programme nicht selbst auf die Chipkarte aufspielen, da dazu sicherkritische Schlüssel benötigt werden. Um die Programme für Chipkarten (Cardlets) wiederum aber nicht zu streng an den Multi–Applikations–Infrastruktur–Betreiber zu koppeln, gibt es eine Vermittlerrolle, den Cardlet–Anbieter. Auch die Relevanz kartenspezifischer Schlüssel ist deutlich geworden. Sollte ein Schlüssel „entwendet“ werden, so ist davon nicht die komplette Charge, sondern nur eine einzelne Karte betroffen.

9.2 Mögliche Weiterentwicklungen

Mögliche Weiterentwicklungen können in vielfältige Richtungen gehen. Durch die Möglichkeit der feingranularen Dezentralisierung des Frameworks und des Prototypen, ist z.B. die Nutzung von Chipkarten mit einem Bastion Host möglich. Dazu muss auf dem Client lediglich das Opencard-Framework und ein kleiner Treiber zur Kommunikation installiert werden.

Auch eine Erweiterung der vorhandenen Campuskarteninfrastruktur wäre denkbar. Momentan werden alle Klassen zur Generierung von Kommandos für Chipkarten auf den Client geladen. Durch die Verlagerung dieser Funktionalität auf den Server könnte der Client schneller und sicherheitstechnisch besser arbeiten, indem er nur noch als Proxy fungiert.

Zum Einsatz des Multi–Applikations–Frameworks für Chipkarten im Rahmen des Campuskartenprojektes müssten von allen ausgegebenen Chipkarten kartenspezifische Schlüssel erfasst und im Kartenschlüssel–Management abgelegt werden. Damit dies effizient geschehen kann, sollten alle prototypisch implementierten Dienste, welche Daten halten, wie das Karten–Status–Abfrage–System, das Kartenschlüssel–Management oder der Cardlet–Anbieter, durch datenbankbasierte Implementierungen ersetzt werden. Eine komfortable Konfiguration dieser Dienste, z.B. über Webseiten, würden das Management des Systems auch ohne tiefer gehende Fachkenntnisse erlauben.

Zur sicheren Umsetzung des Multi–Applikations–Frameworks für Chipkarten sind weiterhin neue Chipkartentypen notwendig, welche erweiterte Sicherheitsmerkmale unterstützen. Zusätzlich müssen die erarbeiteten Dienste mit einem Rechtemanagement gesichert werden.

9.3 Ausblick

Durch die Einführung eines Multi–Applikations–Frameworks, für die Campuskarte der TU–Berlin, ließe sich diese wesentlich kostengünstiger erweitern und anpassen. Die Infrastruktur der Campuskarte ist fertig gestellt, viele Anwendungen werden aber erst in Zukunft entwickelt werden. Zur einfachen und schnellen Implementierung von Anwendungen, welche eine Kartenfunktionalität benötigen, könnte das in

dieser Arbeit beschriebene System wesentlich beitragen. Die technischen Grundlagen sind bereits gegeben, auch wenn die Rekonfiguration der momentan verwendeten Chipkarte, einer G&D Sm@rtCafe Expert 2.0, aus Sicherheitsgründen nur von gesicherten Rechnern stattfinden sollte. Dazu könnten spezielle Terminals bereitgestellt werden, aber auch eine zentrale Stelle, wie die Kartenausgabestelle, ist denkbar. Damit dies funktioniert, muss das vorhandene Karten–Status–Abfrage–System auf die Erfassung aller Chipkarten sofort nach der Lieferung umgestellt werden. Der Lebenszyklus einer jeden Chipkarte müsste mit Zuständen wie „geliefert“ oder „vorpersonalisiert“ gesichert werden. So könnte jede Karte, ob nun Test–, Probe–, Entwickler– oder offizielle Campuskarte sofort eindeutig zugeordnet und eine Rekonfiguration ermöglicht werden.

Meiner Meinung nach werden zukünftige Chipkarten wie kleine, unabhängige Server im Netzwerk agieren können. Es gibt bereits Vorschläge zur direkten Integration von USB–Schnittstellen auf Chipkarten [54], die Integration eines TCP/IP–Stacks dürfte bei ständig steigender Leistung und Speicherkapazität nur eine Frage der Zeit sein. Dem entsprechend unterliegen diese „neuen“ Chipkarten aber auch mindestens den gleichen Kriterien wie herkömmliche sicherheitsrelevante Server in offenen Netzwerken. Sehr wahrscheinlich werden diese neuen Chips nicht nur in Kunststoffkarten, sondern in vielen anderen Geräten verbaut werden, so dass die Bezeichnung „Security-Token“ treffender ist. Die drahtlose Vernetzung wird sich auch in diesem Bereich durchsetzen, so dass zusätzliche Maßnahmen zum Schutz vor Missbrauch geschaffen werden müssen. Verschiedene Token könnten auch mit biometrische Hardware, wie einem Fingerabdruck– oder Irisscanner ausgestattet werden. Sehr wahrscheinlich werden solche Token auch in vorhandene Endgeräte, wie Handys oder PDA's integriert werden.

Die Technik der heutigen Chipkarten erinnert sehr stark an die Anfänge der ersten Personal Computer in den 80'er Jahren, wenngleich mit moderner Software versehen. Die fortschreitende Weiterentwicklung und Miniaturisierung wird uns auch in diesem Bereich noch sehr viele Verbesserungen bringen. Eine sorgfältig und konsequent bedachte Anwendung kann wesentliche Vorteile zum Schutz der Interessen und Rechte eines jeden Einzelnen ermöglichen.

Glossar

3

3DES Erweiterung von ⇒ *DES*, wobei drei Durchläufe mit verschiedenen Schlüsseln durchgeführt werden.

A

AES Symmetrischer Kryptographiealgorithmus. Nachfolger von ⇒ *DES*.

AID Application Identifier, Bezeichner für einen Eintrag im Inhaltsverzeichnis einer ⇒ *Java-Karte*.

APDU Application Protocol Data Unit, Datenstruktur zur Kommunikation zwischen Anwendung und ⇒ *Chipkarte*.

Applet Java-Programm, welches in einem Web–Browser läuft.

Application Initialisierte Programme auf einer ⇒ *VOP* kompatiblen ⇒ *Chipkarte*.

ATR Answer To Reset, Antwort einer ⇒ *Chipkarte* auf das Einlegen in einen ⇒ *Chipkartenleser*.

C

Campuskarte ⇒ *Chipkarte* für Studierende und Mitarbeiter der TU-Berlin.

Card-Manager ⇒ *Cardlet* einer ⇒ *Java-Karte*, welches Kartenhersteller spezifische Zugriffs- und Sicherheitsfunktionen bereitstellt. ⇒ *Security-Domain*.

Cardlet Java-Programm, welches auf einer ⇒ *Java-Karte* läuft.

Chipkarte Allgemeiner Begriff für eine Kunststoffkarte, welche einen oder mehrere Halbleiterchips enthält. ⇒ *Java-Karte*.

Chipkartenleser Gerät, welches einen Einschub für eine Chipkarte bietet, um diese mit dem Computer zu verbinden. Es gibt vier verschiedene Klassen von Chipkartenlesern, Nur Kartenleser (Klasse 1), Kartenleser mit Tastatur (Klasse 2), Kartenleser mit Tastatur und Display (Klasse 3) sowie Kartenleser mit Tastatur, Display und eigenem Zertifikat (Klasse 4).

CPLC Card Production Life Cycle, Beschreibt den Verlauf des Lebenszyklus einer ⇒ *Chipkarte* nach dem ⇒ *VOP*–Standard.

CUID Eindeutige Ziffernfolge zur Kennzeichnung einer ⇒ *Chipkarte*.

D

DES Symmetrischer Kryptographiealgorithmus. ⇒ *AES*.

E

Einweg-Hashfunktion Eine ⇒ *Hashfunktion*, welche sich in eine Richtung sehr leicht, in die andere jedoch nur sehr schwer berechnen läßt. ⇒ *SHA-1, MD5*.

H

Hashfunktion Eine Funktion, welche einen Eingabewert auf einen (im Allgemeinen) kürzeren Ausgabewert abbildet.

J

Java Objektorientierte Programmiersprache und Ausführungsumgebung von Sun Microsystems. ⇒ *Applet, Cardlet, Servlet*.

Java-Karte (engl. Java Card) Chipkarte mit einem Java Betriebssystem. ⇒ *Java*.

K

KDC Karten und ⇒ *Security-Domain* abhängiger spezifischer Schlüssel für den ⇒ *Card-Manager* einer ⇒ *VOP* kompatiblen ⇒ *Chipkarte*.

KMC Cargenabhängiger Hauptschlüssel für den ⇒ *Card-Manager* einer ⇒ *VOP* kompatiblen ⇒ *Chipkarte*.

L

Loadfile Auf einer ⇒ *VOP* kompatiblen ⇒ *Chipkarte* vorhandene Dateien.

M

MAFC Abkürzung für Multi-Applikations-Framework für Chipkarten.

MD5 Eine ⇒ *Einweg-Hashfunktion*, welche einen Eingabewert auf einen 128-Bit Ausgabewert abbildet.

R

RSA Asymmetrischer Kryptographiealgorithmus.

S

Security-Domain ⇒ *Cardlet* einer ⇒ *Java-Karte*, welche Applikations spezifische Zugriffs- und Sicherheitsfunktionen bereitstellt. ⇒ *Card-Manager*.

Servlet Java-Programm, welches auf einem Web–Server läuft.

SHA-1 Eine ⇒ *Einweg-Hashfunktion*, welche einen Eingabewert auf einen 160-Bit Ausgabewert abbildet.

Sitzung (engl. Session) Zeitspanne zwischen An- und Abmeldung eines Dienstes oder einer Verbindung, während der sämtliche Kommunikation und notwendige Berechnungen ablaufen.

T

Terminal Benutzungsschnittstelle für einen Computer.

U

UML Unified Modeling Language, Modellierungssprache für IT-bezogene Sachverhalte.

V

VOP Visa Open Platform, Spezifikationen zum Management einer ⇒ *Chipkarte*.

Z

Zustand (engl. State) Vorgabe während des Lebens eines Objektes oder einer Interaktion, während diese(s) eine Bedingung erfüllt, Aktionen ausführt oder auf ein Ereignis wartet. Übersetzt aus [33].

Zustandsobjekt Ein Objekt, welches einen Zustands repräsentiert. Zusätzlich besitzt es Methoden zur Ausführung von Aktionen. ⇒ *Zustand*.

Literaturverzeichnis

Die angegebenen Internetadressen entsprechen dem Stand vom 14. Dezember 2003.

[1] *Microsoft .NET Developer Homepage*, 2003. URL `http://msdn.microsoft.com/netframework/`.

[2] *SUN Java Homepage*, 2003. URL `http://java.sun.com`.

[3] *SUN Java2 Micro Edition Homepage*, 2003. URL `http://java.sun.com/j2me/`.

[4] *Open Card Homepage*, 2003. URL `http://www.opencard.org`.

[5] *Tomcat Homepage*, 2003. URL `http://jakarta.apache.org/tomcat/`.

[6] *Concurrent Versions System Homepage*, 2003. URL `http://www.cvshome.org`.

[7] *CVSNT Homepage*, 2003. URL `http://www.cvsnt.org`.

[8] *WinCVS Homepage*, 2003. URL `http://www.wincvs.org`.

[9] *Rahmen–Pflichtenheft — Chipkartenbasierte Dienstleistungssysteme an den Berliner und Brandenburger Hochschulen Version 2.0.* Arbeitskreis Campuskarte der Berliner und Brandenburger Hochschulen, 2000.

[10] *RFC 2821, Simple Mail Transfer Protocol.* AT&T Laboratories, 2001.

[11] *RFC 2246, The TLS Protocol Version 1.0.* Certicom, 1999.

[12] Zhiqun Chen. *Java Card Technology for Smart Cards. Architecture und Programmer's Guide.* Sun Microsystems Inc., Palo Alto, 2000.

[13] *RFC 1939, Post Office Protocol - Version 3.* Dover Beach Consulting, Inc., 1996.

[14] Bruce Eckel. *Thinking in Java.* Prentice Hall PTR, New Jersey, 2002.

[15] Peter Rechenberg et. Al. *Informatik-Handbuch.* Carl Hanser Verlag, München, 1997.

[16] *GemXpresso 211 V2 Card Reference Manual Version 2.0.* Gemplus, 2000.

[17] *GemXpresso Pro R3 Card Reference Manual.* Gemplus, 2001.

[18] *Reference Manual Sm@rtCafe Expert 2.0 Edition 03/03.* Giesecke & Devrient GmbH, 2003.

[19] *Open Platform Card Specification, Version 2.0.1.* Global Platform Inc., 2000.

[20] *Global Platform Card Specification, Version 2.1.1.* Global Platform Inc., 2003.

[21] Uwe Hansmann. *Smart Card Application Development Using Java.* Springer, Berlin, 2000.

[22] Thomas Hildmann. *Vermeidung von Datenspuren bei smartcardbasierten Authentisierungssystemen. In Verlässliche IT–System 2001, Sicherheit in komplexen IT-Infrastrukturen.* Wiesbaden, 2001.

[23] *RFC 791, Internet Protocol.* Information Sciences Institute at University of Southern California, 1981.

[24] *RFC 1035, Domain Names.* ISI, 1987.

[25] *RFC 854, Telnet Protocol.* ISI, 1983.

[26] *RFC 959, File Transport Protocol.* ISI, 1985.

[27] *RFC 1321, The MD5 Message-Digest Algorithm.* MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992.

[28] *RFC 2068, Hypertext Transfer Protocol 1.1.* MIT/LCS, 1997.

[29] *Advanced Encryption Standard (AES).* National Institute of Standards and Technology, 2001. URL `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[30] *Data Encryption Standard (DES).* National Institute of Standards and Technology, 1993. URL `http://www.itl.nist.gov/fipspubs/fip46-2.htm`.

[31] *Secure Hash Standard.* National Institute of Standards and Technology, 1995. URL `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[32] *Common Object Request Broker Architecture: Core Specification Version 3.0.* Object Management Group, Inc, 2002.

[33] *OMG Unified Modeling Language Specification Version 1.5.* Object Management Group, Inc, 2003.

[34] Andreas Pfitzmann. *Script: Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme.* TU-Dresden, Fakultät Informatik, Dresden, 2000.

[35] *Dokumentation des Cardlets Campuskarte.* PRZ/TU-Berlin, 2002.

[36] *Dokumentation Java-Integration Campuskarte.* PRZ/TU-Berlin, 2003.

[37] *RFC 2822, Internet Message Format.* QUALCOMM Incorporated, 2001.

[38] *RFC 2818, HTTP Over TLS.* RTFM, Inc., 2000.

[39] Bruce Schneier. *Angewandte Kryptographie.* Addison-Wesley, Bonn, 1996.

[40] Bruce Schneier. *Secrets & Lies: IT-Sicherheit in einer vernetzten Welt.* dpunkt.verlag, Heidelberg, 2001.

[41] Dehla Sokenou. *Vorlesungsfolien: Objektorientierte Softwareentwicklung SS2002.* Berlin, 2002.

[42] William Stallings. *Data und Computer Communication.* Prentice Hall, New Jersey, 1999.

[43] *Java Card 2.1 Application Programming Interface.* Sun Microsystems Inc., 1999.

[44] Jan Bosch u.A. *„Object-Oriented Frameworks - Problems & Experiences“Artikel in Building Application Frameworks.* John Wiley & Sons, 1999.

[45] R.L. Rivest und A. Shamir und L.M. Adleman. A method for obtaining digital signatures und public-key cryptosystems. *Communications of the ACM*, Februar 1978.

[46] Cay S. Horstmann und Gary Cornell. *Core Java*, volume I. Sun Microsystems Inc., Palo Alto, 2002.

[47] Cay S. Horstmann und Gary Cornell. *Core Java*, volume II. Sun Microsystems Inc., Palo Alto, 2002.

[48] Andreas Pfitzmann und Gritta Wolf. Charakteristika von Schutzzielen und Konsequenzen für Benutzungsschnittstellen. *Informatik Spektrum*, Juni 2000.

[49] Grady Booch und James Rumbaugh und Ivar Jacobsen. *The Unified Modeling Language User Guide.* Addison-Wesley, Boston, 1999.

[50] Thomas Hildmann und Jörg Bartholdt. *Managing Trust between collaborating Companies using outsourced Role Based Access Control.* Fifth ACM Workshop on Role-Based Access Control, 1999.

[51] Stephen Stelting und Olav Maassen. *Applied Java Patterns.* Sun Microsystems Inc., Palo Alto, 2002.

[52] Johannes Link und Peter Fröhlich. *Unit Tests mit Java.* Dpunkt Verlag, Heidelberg, 2002.

[53] Margot Bittner und Wilfried Koch. *Script: Objektorientierte Analyse und Design, Die Fusion Methode unter Verwendung von UML.* TU-Berlin, Berlin, 2001.

[54] Wolfgang Rankl und Wolfgang Effing. *Handbuch der Chipkarten. Aufbau - Funktionsweise - Einsatz von Smart Cards, 4. Auflage.* Hanser, München, 2002.

[55] *RFC 2060, Internet Message Access Protocol - version 4rev1.* University of Washington, 1996.

[56] *Open Platform Card Production Guide, Version 2.02.* Visa International, 2000.

[57] Thomas J. Wilke. *Diplomarbeit: Konzeption eines modularen und vernetzten Zugangskontrollsystems auf Basis der Produktfamilie SIPORT.* Universität Karlsruhe, 1996.

[58] Adam Wolisz. *Script: Telecommunication Networks WS2000/01.* TU-Berlin, Berlin, 2000/2001.

[59] *Simple Object Access Protocol.* World Wide Web Consortium (W3C), 2000. URL `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.