Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme
Harburger Schloßstraße 20
21079 Hamburg

Ein objektorientiertes Softwaremodell zur Benutzung von Chipkarten in persistenten und verteilten Systemen

Diplomarbeit
Studiengang Elektrotechnik
TU Hamburg-Harburg

Betreuer: Florian Matthes
Autor: Steffen Sperling
Sperling@tu-harburg.de

31. Juli 1998

Firmen- und Produktnamen werden in der gesamten Arbeit in GROSSBUCHSTABEN gesetzt, Programmfragmente in `Schreibmaschinenschrift`.
Wo es nicht oder nur über weniger prägnante Formulierungen möglich war, einen englischen Ausdruck zu ersetzten, wird dieser *kursiv* gesetzt.

Zusammenfassung

Im Rahmen dieser Arbeit wird ein objektorientiertes Modell zur Benutzung von Chipkarten entwickelt, das die Nutzung von Chipkarten in persistenten und verteilten Systemen ermöglicht.
Die Modellierung der Funktionen von Chipkarten wird mit der *Unified Modeling Language* nach Standardverfahren der objektorientierten Softwareanalyse und -entwicklung durchgeführt. Als Modell für die Chipkartenfunktionen wird die Dienstleistung verwendet. Dienste werden einem System, das Chipkarten nutzt, durch einen Diensttreiber verfügbar gemacht. Die Modellierung der Dienste orientiert sich an den Anforderungen von Anwendungen und den Normen für Chipkarten. Entsprechend den Anwendungfällen für die Nutzung von Chipkarten werden Diensttreiber entworfen. Anwendungsfälle sind für Benutzer und Herausgeber einer Karte verschieden. Für den Herausgeber der Karte werden Diensttreiber für die Anwendungsfälle betrachtet, die für die Personalisierung von Chipkarten erforderlich sind. Die Personalisierung des Dateisystems, die Verwaltung von Sicherheitscodes und das irreversible Abschließen der Personalisierung sind Anwendungsfälle für den Herausgeber der Karte. Der Benutzer der Chipkarte benötigt Dienste der Karte, die Zugriff auf Daten, Sicherheitscodes und Funktionen der Karte ermöglichen. Die diesen Anwendungsfällen entsprechenden Dienste sind: Personalisierungsdienst, Dateisystemdienst, PIN-Verwaltungsdienst und Verschlüsselungsdienst.
Standardverfahren der objektorientierten Softwareentwicklung werden zur Modellierung und Implementierung verwendet. Softwareentwicklung mit Chipkarten beinhaltet hardwarenahen Programmentwurf. Durch die Chipkartenhardware bedingte Unterschiede bei der Vorgehensweise werden berücksichtigt.
Eine Beispielanwendung für Chipkarten wird mit Verfahren der objektorientierten Analyse entworfen. Für diese Anwendung werden Treiber für die Chipkartendienste einer konkreten Karte, der DELARUE DXPLUS, entwickelt: Personalisierungsdienst, Dateisystemdienst, Verschlüsselungsdienst und PIN-Verwaltungsdienst. Um mit der Chipkarte zu kommunizieren, wird ein Treiber für das PHILIPS PE 122-210-Chipkartenschreiblesegerät entwickelt.
Die Klassen sind in JAVA implementiert worden. Der Test ist mit dem OPENCARD-System, um Chipkarten anzusprechen, durchgeführt worden. Die entworfenen Programme arbeiten zuverlässig.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Kunststoffkarten zur bargeldlosen Bezahlung wurden zum ersten Mal 1950 vom Diners Club in den Vereinigten Staaten als überregionales Zahlungsmittel ausgegeben. Die Funktion der Karten war zu dieser Zeit sehr einfach. Anhand des in den Kunststoff eingeprägten Namens des Benutzers und der Kartennummer konnten durch ein mechanisches Verfahren die Daten auf Rechnungsformulare übertragen werden. Die Echtheit der Karte wurde vom Kassenpersonal anhand optischer Merkmale des Kunstoffkörpers überprüft.

Da solche Karten nicht maschinenlesbar sind, werden sie durch einen auf dem Kunststoffkörper angebrachten Magnetstreifen ergänzt. Auf diesem Magnetstreifen kann maschinenlesbare digitale Information abgelegt werden. Der Nachteil solcher Magnetstreifenkarten ist, daß die gespeicherten Daten einfach les- und veränderbar sind. Geheime Information kann auf diesen Karten nicht sicher abgelegt werden. Zur Sicherstellung der Echtheit der Daten ist zusätzlicher Aufwand nötig. Diese Überprüfung wird häufig durch eine direkte Verbindung mit einem Zentralrechner gelöst.

1968 wurden erste Patente für Kunststoffkarten mit eingebettetem Mikroprozessor angemeldet. Anfang der 1980'er Jahre war es technisch möglich, Mikroprozessoren zu einem für Massenanwendungen geeigneten Preis in Kunststoffkarten einzubetten. 1984 wurden von der französischen Telefongesellschaft PTT Chipkarten als bargeldloses Zahlungsmittel in öffentlichen Telefonzellen eingeführt. Die Deutsche Telekom folgte 1985.

Das 1994 eingeführte, globale, digitale Mobiltelefonnetz GSM (Groupe Spécial Mobile) verwendet Chipkarten zur Freischaltung der Mobiltelefone.

1996 wurde für alle gesetzlich Krankenversicherten in Deutschland eine Chipkarte anstelle der vorher üblichen Krankenscheine eingeführt. Die erste Bankkarte mit Mikroprozessor wurde in Deutschland 1997 mit der Geldkarte eingeführt.

Neben solchen Massenanwendungen existieren spezialisierte Anwendungsbereiche wie Zugangskontrollsysteme und Zeiterfassungssysteme mit Chipkarten.

Häufig werden Chipkarten in komplexen verteilten Softwaresystemen eingesetzt. Vom Benutzer veranlaßte Aktionen werden mit Hilfe einer Chipkarte authentifiziert. In anderen Kartenanwendungen wird von einem selbständig arbeitenden Schreiblesegerät ohne Anschluß an weitere Systeme auf Chipkarten zugegriffen. Das Schreiblesegerät führt alle benötigten Aktionen autonom aus.

Der Kunststoffkörper der Karte enthält einen eingebauten Mikroprozessor, der in einem eingebetteten nichtflüchtigen Speicher Daten speichern und Funktionen ausführen kann. Diese Möglichkeiten werden dazu genutzt, Daten vor unerlaubtem Zugriff zu schützen. Das exter-

ne System überprüft die Echtheit der Chipkarte mit Hilfe einer kryptographischen Funktion der Chipkarte. Die Karte wird authentifiziert und es werden ausschließlich vertrauenswürdigen Partnern Daten übermittelt. Das externe System seinerseits bekommt keinen Zugriff auf geschützte Informationen der Karte, wenn es sich nicht gegenüber der Karte authentifiziert hat. Durch das aktive Sicherheitssystem und die Mobilität der Karte sind Chipkarten für Anwendungen mit hohem Daten-Sicherheitsniveau geeignet.

1.1 Motivation

Physikalische Abmessungen, der Ort der Kontakte und die zum Aktivieren der Chipkarte notwendigen elektrischen Signale sind durch eine internationale Norm festgelegt. Die physikalische Datenübertragung und die Datenaustauschprotokolle sind ebenfalls standardisiert. Ein Teil der Befehle, die eine Karte ausführen kann, ist durch eine internationale Norm festgelegt. Obwohl auf Hardwareseite international anerkannte Normen für Chipkarten verwendet werden, sind die Treiber für den Zugriff auf Chipkarten und Chipkartenschreiblesegeräte nicht standardisiert. Softwaresysteme, die mit Chipkarten arbeiten, sind daher schwer pfleg- und erweiterbar. Ein Wechsel von einem Kartentyp auf eine leistungsfähigere Karte zieht nicht nur Modifikationen an der Software, um die Karte anzusprechen, nach sich, sondern auch Modifikationen an der Hardware, da ein Teil der Anwendung fest ins Schreiblesegerät eingebaut ist. Um auf Karten von verschiedenen Herstellern zuzugreifen, existieren auf derselben Plattform unterschiedliche Schnittstellen. Um Chipkarten auf unterschiedlichen Plattformen zu verwenden, nutzen verschiedene Anwendungen unterschiedliche Programmiermodelle und Softwareschnittstellen.

1.2 Zielsetzung der Arbeit

In dieser Arbeit wird anhand einer Anwendungsfallanalyse, vom ISO-Standard für Chipkarten ausgehend, ein objektorientiertes Softwaremodell zur Benutzung von Chipkarten in persistenten und verteilten Systemen entworfen.
Persistente Systeme ermöglichen die langlebige und sichere Speicherung großer Mengen von Daten unterschiedlicher Art. Merkmale solcher Systeme sind (nach [Matthes 93]).

- **Persistente Datenspeicherung:** Komplex strukturierte, langlebige Daten, die von mehreren Programmen genutzt werden, werden gemeinsam nutz- und modifizierbar gespeichert.
- **Generische Programmierung:** Stereotype Programmteile wie Konsistenz- und Sicherheitsüberprüfungen werden aus parametrisierbaren hochsprachlichen Beschreibungen abgeleitet.
- **Externe Kommunikation:** Auf Objekte innerhalb des Systems kann aus anderen Systemen nur über wohldefinierte Schnittstellen zugegriffen werden; umgekehrt können Funktionen des Systems extern angestoßen werden, um Aktionen basierend auf den Daten des Systems zu steuern.

Der Aspekt der externen Kommunikation beinhaltet Möglichkeiten, Transaktionen durchzuführen. Ein Beispiel ist das Abbuchen von Guthaben eines Kontos am Geldautomaten. Für

solche Transaktionen ist es erforderlich, Daten des Benutzers elektronisch verfügbar zu haben. Das System, mit dem die Transaktion durchgeführt wird, kann anhand dieser Daten den Benutzer authentifizieren und Transaktionen ermöglichen. Zur Speicherung solcher Daten werden Chipkarten verwendet.
Verteilte Systeme sind Systeme aus physikalisch verteilten Computern, die durch ein Netzwerk verbunden und mit verteilter Software ausgestattet sind. Die Schnittstellen zum Funktionsaufruf auf anderen Computern des gleichen Systems sind mit den internen Schnittstellen identisch. Für den Benutzer ist nicht sichtbar, daß es sich um verschiedende Systeme handelt und daß zur Abarbeitung des gegebenen Auftrags mehrere Systeme aktiv werden können (nach [Coulouris et al. 94]). Verteilte Systeme sind durch ubiquitäre Betriebsmittel gekennzeichnet. Beispiele für solche Betriebsmittel sind Geldauswurfsmechanismen, Tastaturen zur Eingabe eines Geheimcodes und Magnetstreifenleser in Geldautomaten. Die Freigabe der Betriebsmittel in verteilten Anwendungen kann mit Hilfe von Chipkarten erlangt werden. Das System kann ohne Zugriff auf zentrale Datenbanken die Echtheit der Chipkarte anhand kryptographischer Verfahren überprüfen.
Das bekannteste Beispiel für ein verteiltes System, das mit Chipkarten arbeitet, ist das öffentliche Telefonsystem der Deutschen Telekom. Das Kartentelefon kann anhand der Karte ohne Zugriff auf das Hintergrundsystem das Guthaben des Benutzers überprüfen. Der Benutzer kann, solange Guthaben auf der Karte vorhanden ist, über die an allen Telefonen identische Benutzerschnittstelle auf das Telefonnetz zugreifen. Das Kartentelefon tätigt dabei die Transaktion mit der Chipkarte, indem Geldeinheiten abgebucht werden.

1.3 Aufgabenstellung

Es wird mit Hilfe von Standardverfahren der objektorientierten Modellierung ein Modell entworfen, das die Verwendung von Chipkarten in persistenten und verteilten Systemen ermöglicht. Leitlinien für die Entwicklung des Softwaremodells sind:

- **Objektorientierte Modellierung:** Das Modell wird objektorientiert entwickelt, die Modellierung erfolgt mit Hilfe der *Unified Modeling Language* (UML) [Fowler 97]. Die hardwarenahen Aufrufe von Kartenfunktionen werden so gekapselt, daß sie objektorientiert verwendet werden können.

- **Verwendbarkeit von Chipkarten in persistenten und verteilten Systemen:** Die Benutzung von Chipkarten in persistenten und verteilten Systemen (siehe oben) wird durch die Implementierung ermöglicht.

- **Plattformunabhängigkeit:** Es wird plattformunabhängig modelliert und in einer plattformunabhängigen, objektorientierten Programmiersprache implementiert.

- Berücksichtigung des **Persistenzaspekts** der Chipkarte: Chipkarten speichern Daten und ihren Zustand auch nach dem Abschalten der Versorgungsspannung.

- Berücksichtigung der physikalischen **Mobilität** der Karte: Chipkarten können an verschiedene Systeme angekoppelt werden.

- Verwendung vorhandener **Normen**: Normen und Ansätze zur Verwendung von Chipkarten [ISO 94; Herrmann, Husemann 98; Sun 97] werden ausgenutzt, um Kompatibilität mit bestehenden Systemen zu gewährleisten.

- **Unabhängigkeit von der verwendeten Chipkarte:** Ähnliche Funktionen verschiedener Chipkarten werden über gleiche Schnittstellen angesprochen. Das Modell wird so entwickelt, daß darauf aufsetzende Anwendungen Chipkarten unterschiedlicher Hersteller mit unterschiedlichen Befehlssätzen nutzen können.

- Entwurf einer **Anwendung** mit einem **objektorientierten Entwurfsverfahren:** Eine Beispielanwendung wird objektorientiert entworfen und implementiert. Die durch die Verwendung von Chipkartenhardware bedingten Unterschiede in der Vorgehensweise beim Entwurf der Anwendung werden aufgezeigt. Die Anwendung wird mit einer konkreten Chipkarte entworfen. Die Implementierung wird in der Programmiersprache JAVA [Eckel 98] vorgenommen.

Kapitel 2

Grundlagen der Chipkartentechnik

Chipkarten haben das Format einer Kredit- oder Eurocheque-Karte. Durch einen in den Kunststoffkörper eingebetteten Mikroprozessor wird die geringe Speicherkapazität von ca. 1000 Bits auf Magnetstreifenkarten auf bis zu 16 KBytes erhöht. Der eingebettete Mikrocontroller dieser Karten enthält einen Prozessorkern, RAM, ROM und EEPROM (*electrically erasable programmable read only memory*). Zugriff auf Daten in Chipkarten wird, im Gegensatz zu magneto-elektrischen Verfahren bei Magnetstreifenkarten, über elektronische Verfahren erlangt.
Die Daten werden elektronisch vor unerlaubtem Zugriff geschützt. Die Karte hat gegenüber empfindlichen Magnetstreifenkarten eine deutlich erhöhte Lebensdauer. Eine Chipkarte erlaubt 10.000 Steckzyklen, eine Magnetstreifenkarte 1000 Steckzyklen, bis sie mechanisch verbraucht ist.
Das Betriebssystem der Karte kontrolliert den Zugriff auf Daten im EEPROM des Kartenprozessors. Die Daten sind im EEPROM als Dateien angeordnet. Dateien können mit einem Zugriffscode (persönliche Identifikationsnummer, PIN), den der Prozessor überprüft, geschützt werden. Die Karte kann sich nach dem Präsentieren von falschen Codes sperren und so weiteren Zugriff auf gespeicherte Daten und Funktionen der Karte verhindern.
Ein Überblick über mechanische und elektrische Eigenschaften von Chipkarten sowie deren Betriebssysteme wird in Abschnitt 2.1 gegeben. Zunächst werden die für Chipkarten verwendeten Normen erläutert (vergleiche Abschnitt 2.1.1), dann werden die Charakteristika der Betriebssysteme betrachtet (siehe Abschnitt 2.1.3).
Moderne Karten verfügen über die Möglichkeit einen kryptographischen Algorithmus auszuführen und so Daten zu ver- oder entschlüsseln (siehe Abschnitt 2.1.6).
Chipkarten werden für verschiedene Anwendungen hergestellt (vergleiche Abschnitt 2.1.7). Die DXPLUS Chipkarte des Herstellers DeLaRue, wird in Abschnitt 2.1.9 detailliert erläutert.
Eine vollständige Beschreibung technischer Details ist in [Wolfgang Rankl 96] zu finden.

Um in Softwaresystemen Chipkarten zu verwenden, muß von einer Programmiersprache auf sie zugegriffen werden. Die plattformunabhängig ausführbare, objektorientierte Programmiersprache JAVA (vergleiche hierzu Abschnitt 2.2.1) wird im Rahmen dieser Arbeit zur Implementierung der entworfenen Modelle verwendet.
Mit JAVA sind Möglichkeiten vorhanden, Chipkarten und Kartenschreiblesegeräte anzusprechen: Das Paket `javax.smartcard` des Herstellers SUN (siehe Abschnitt 2.2.2) und das OPENCARD-Framework (vergleiche Abschnitt 2.2.2), das ein Zusammenschluß aus Chipkarten-

herstellern entworfen hat.

2.1 Eigenschaften von Chipkarten

2.1.1 Normung

Die Kommunikation zwischen Karte und Schreiblesegerät spezifiziert eine internationale Normen (ISO 7816 1–3). Diese Norm legt physikalische Daten und das Kommunikationsprotokoll von Chipkarten fest. Die Befehle, die eine Chipkarte ausführen kann, und die Datenspeicherung in Dateien sind in einer Erweiterung dieser Norm (ISO 7816 Teil 4) festgelegt.
ISO 7816 ist in 4 Teile gegliedert:

ISO 7816 Teil 1: Physikalische Spezifikation [ISO 87]

ISO 7816 Teil 2: Dimension und Ort der Kontakte [ISO 88]

ISO 7816 Teil 3: Elektrische Signale und Übertragungsprotokolle [ISO 89]

ISO 7816 Teil 4: Herstellerübergreifende Befehle (Dateistrukturen, Befehle für allgemeine Anwendungen) [ISO 94]

Physikalische Daten

Abmessungen: Der erste Teil der ISO Norm spezifiziert die Abmessungen des Kunststoffkörpers der Karte (siehe Abbildung 2.1) und das Verhalten bei mechanischer Beanspruchung. Die Position der Kontakte und deren Zuordnung zu den in Teil 3 festgelegten Signalen sind

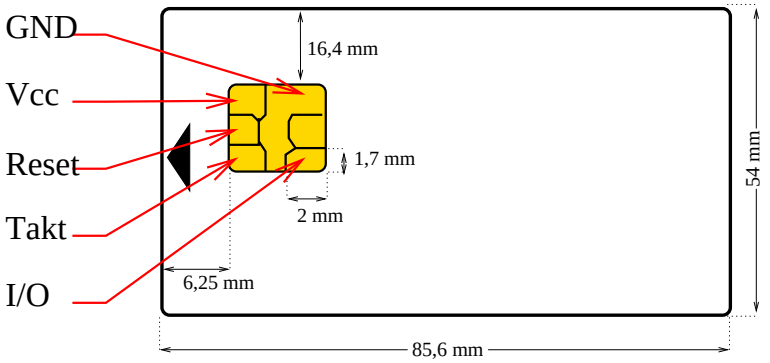

Abbildung 2.1: Chipkarte nach ISO-Norm

in Teil 2 der ISO-Norm festgelegt (Abbildung 2.1). Das am häufigsten eingesetzte Format für Chipkarten ist das in der ISO-Norm mit ID-1 bezeichnete Format: Die Karte hat eine Höhe von 54 mm, ein Breite von 85.6 mm und eine Dicke von 0,76 mm. Das Kontaktfeld hat einen Abstand von 16,4 mm von der oberen Kante der Karte und von 6.25 mm von der linken Kante. Die Kontakte sind minimal 2 mm breit und 1.7 mm hoch. Die beiden Reihen von Kontakten haben einen Abstand von 5,5 mm. Für spezielle Anwendungen (z.B. Mobiltelefone) werden kleinere Karten verwendet, deren Abmessungen ebenfalls festgelegt sind.

Kontakte: Chipkarten stellen über fünf Kontakte die Verbindung zum Schreiblesegerät her:

- Versorgungsspannung (zwischen 3 und 5 Volt),
- Masse (GND),
- Takt (3,57 MHz nach ISO),
- Reset (um die Karte in einen definierten Zustand zu versetzen) und
- ein Kontakt zur bidirektionalen, halbduplexen Datenübertragung.

Ein 6. Kontakt wurde früher verwendet, um der Karte Schreibspannung für das EEPROM zur Verfügung zu stellen. Heutige Karten verfügen über eine interne Ladungspumpe, um diese Spannung aus der Versorgungsspannung zu generieren.

Physikalische Datenübertragung: Um mit dem Schreiblesegerät zu kommunizieren, steht der Chipkarte eine bidirektionale, halbduplexe serielle Schnittstelle zur Verfügung, die bei einem Takt von 3,57 MHz am Takteingang mit einer Bitrate von 9600 Bit/s arbeitet. Die Dauer, während der ein Bit anliegen soll (*elementary time unit*, ETU), ist dabei 1/Bitrate. Die Norm legt fest, daß der Pegel auf der Leitung von 0,3 ETU bis 0,7 ETU nach Beginn des Zeitschlitzes dem vorgegebenen Pegel entsprechen muß. Ein Datenbyte besteht aus 12 Bit: 1 Startbit, 8 Datenbit, 1 Paritätsbit (gerade Parität wird verwendet) und 2 Stoppbits die die Wartezeit zwischen zwei gesendeten Datenbytes (*guardtime*) bilden (Abbildung 2.2). Chipkarten können den *high*-Zustand der Leitung als gesetztes Bit (*direct convention*) interpretieren, oder den *low*-Zustand (*inverse convention*). Welches von beiden Formaten verwendet wird, wird an der Antwort auf einen Resetpuls erkannt (siehe Abschnitt 2.1.1).

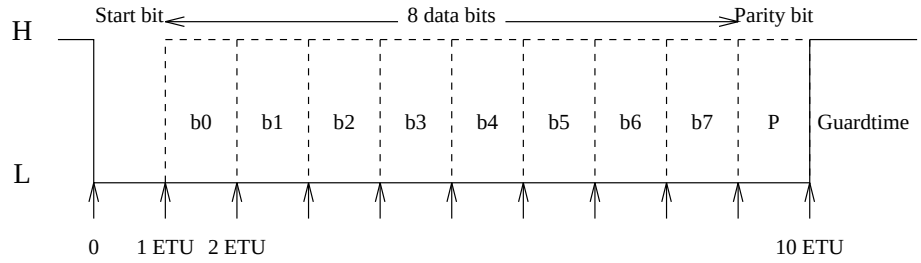

Abbildung 2.2: Ein Datenbyte im ISO Format

Protokoll zur Datenübertragung

Fehlererkennungsprotokoll: Es sind zwei verschiedene Verfahren zur Fehlererkennung in der Norm standardisiert. Das erste Verfahren ergänzt jedes übertragene Byte um ein Paritätsbit (siehe Abbildung 2.2). Wenn nicht gesendet wird, befindet sich die Leitung auf *high*-Pegel: Der Empfänger des letzten Bytes überprüft das Paritätsbit und kann durch einen *low*-Pegel auf der Leitung während der Wartezeit das zuletzt gesendete Byte nochmals vom Sender anfordern. Das geschilderte Fehlererkennungsprotokoll wird, da es das erste implementierte Verfahren ist, $T = 0$ Protokoll genannt.

Das zweite verwendete Verfahren ist blockorientiert: Es wird eine Prüfsumme über den gesamten Befehlsblock gebildet und dem Befehlsblock hinzugefügt. Das Verfahren wird mit $T = 1$ bezeichnet. Bei falscher Prüfsumme wird der gesamte Befehlsblock erneut übertragen.
Welches Fehlerkorrekturverfahren benutzt wird, erkennt das Schreiblesegerät an der Antwort auf einen Resetpuls; die Fehlererkennung wird vom Schreiblesegerät darauf angepaßt.

Antwort auf einen Resetpuls: Sowohl wenn Versorgungsspannung an die Karte angelegt wird als auch nach Empfang eines Resetpulses, initialisiert sich der Prozessor auf der Karte und führt einen Selbsttest durch. Danach wird eine *answer to reset* (ATR) Bytefolge an das Schreiblesegerät gesendet. An dieser Bytefolge erkennt das Schreiblesegerät, in welchem Modus (*direct/inverse convention*) gearbeitet wird, das Fehlererkennungsprotokoll, den Kartentyp, sowie den internen Status der Karte nach dem Selbsttest.

Name	obligatorisch	Bedeutung
TS	ja	$3B_{hex} \Rightarrow$ direct convention $3F_{hex} \Rightarrow$ inverse convention
T0	ja	Formatbyte Bits 1–4 $\Rightarrow$ Anzahl der herstellerspezifischen Bytes Bits 5–8 $\Rightarrow$ TA1...TD1 werden übertragen
TA1	nein	Übertragungsfrequenzteiler
TB1	nein	Programmierspannung bei Schreibbefehlen (in V)
TC1	nein	zusätzliche Wartezeit zwischen zwei Bytes (in ETU)
TD1	nein	Bits 1–4 zeigen das verwendete Protokoll an ($0000_{bin} \Rightarrow T = 0, \quad 0001_{bin} \Rightarrow T = 1$) Bits 5–8 zeigen an, ob TA2..TD2 übertragen werden, Codierung wie T0
TA2-TD2	nein	weitere Protokollparameter
weitere	nein	herstellerspezifische Bytes

Tabelle 2.1: Übersicht über den Aufbau der *answer to reset*

Ein Teil der gesendeten Bytes ist obligatorisch, andere können weggelassen werden (vergleiche Tabelle 2.1). Fehlt ein Byte, so werden für den entsprechenden Parameter Standardwerte angenommen.
Detailliertere Beschreibungen der ATR-Sequenzen finden sich in den Handbüchern der Karten [DeL 98a; DeL 98b] und der ISO-Norm [ISO 89].

Übertragungsprotokoll: Es wird ein *challenge-response*-Protokoll zum Datenaustausch verwendet. Die Karte sendet nur dann, wenn vorher vom Schreiblesegerät ein Befehl empfangen wird. Über die bidirektionale Schnittstelle tauscht das *card operating system* (COS) der Karte mit einem standardisierten Protokoll Daten aus: Es werden jeweils 5 Bytes Befehlscode plus maximal 255 Bytes Parameter von der Karte erwartet. Daraufhin wird eine Antwort von minimal 2 bis maximal 257 Bytes von der Karte gesendet. Ein Befehlsblock (*application protocol data unit*, APDU) setzt sich aus der Instruktionsklasse (CLA), dem Instruktionsbyte

(INS), zwei Parameterbytes (P1 und P2) und einem Längenbyte (LEN) zusammen (siehe Abbildung 2.3). Es folgt die im LEN-Feld angegebene Anzahl an Bytes bei Schreibbefehlen. Bei Lesebefehlen sendet die Karte die im LEN-Feld angegebene Anzahl von Bytes an das Schreiblesegerät. Das CLA-Byte kennzeichnet das Einsatzgebiet der Chipkarte (siehe Tabelle 2.2). Die

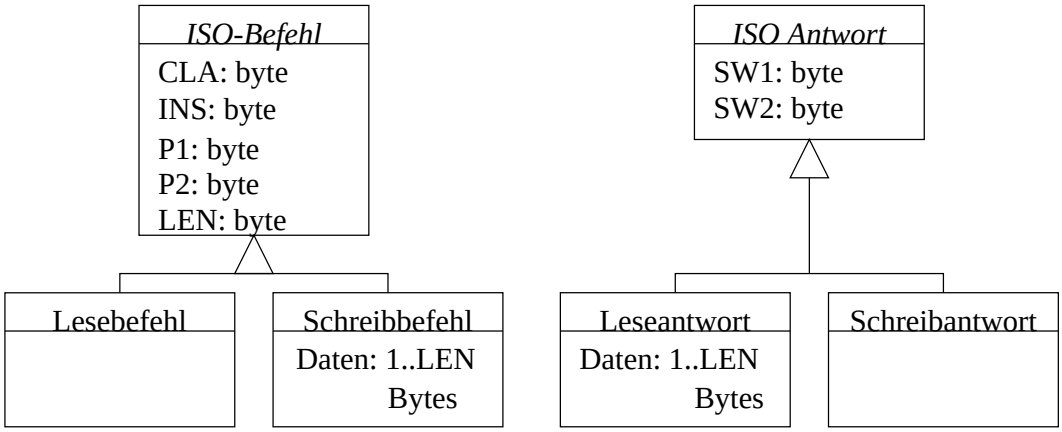

Abbildung 2.3: ISO 7816 Befehle

beiden Parameterbytes P1 und P2 werden verwendet, um indiziert auf den Kartenspeicher zuzugreifen.

CLA-Byte	Einsatzgebiet
0	Multifunktionskarten nach ISO 7816-4
$8X_{hex}$	Kreditkarten mit Chip
$A0_{hex}$	GSM Mobiltelefonkarten
$D0_{hex}$	Dallas iButton

Tabelle 2.2: CLA-Bytes und zugehörige Anwendungen

Dateisystem

Eine Chipkarte verfügt über ein 1 bis 16 Kilobytes großes EEPROM, in dem Benutzer- und Herstellerdaten les- und schreibbar abgelegt werden. Das EEPROM kann nur über das COS der Karte gelesen oder geschrieben werden. Die Adreßleitungen des Kartenprozessors sind nicht von außen zugänglich. Daten im EEPROM werden in Verzeichnisse und Dateien strukturiert.
In Teil 4 der ISO Norm 7816 werden das Dateisystem und die Befehle für den Dateizugriff spezifiziert. Die meisten Kartenhersteller verwenden eine Untermenge der standardisierten Funktionen, die auf die spezielle Anwendung der Karte abgestimmt ist.
Funktionen zur Erzeugung und Veränderung von Dateien und Verzeichnissen sind vorhanden. Die Verzeichnisstruktur in der Karte besteht aus einem Hauptverzeichnis, *masterfile* (Abkürzung MF) genannt, das mehrere einfache Dateien (*elementary files*, Abkürzung EF) und Verzeichnisse (*dedicated files*, Abkürzung DF) enthalten kann.

Verzeichnisse sind Dateien, die bei ihrer Erzeugung als *dedicated files* gekennzeichnet werden. Die Anzahl der möglichen Unterverzeichnisebenen ist von Karte zu Karte unterschiedlich. Es existieren Karten mit einer Verzeichnisebene, andere erlauben beliebig tiefe Schachtelung, die nur durch die Speicherkapazität des EEPROMs beschränkt ist.
Dateien bestehen aus einem Dateikopf und einem Speicherbereich fester Länge, der binäre Daten enthält. Der Dateikopf wird im Datenbereich der zugehörigen Verzeichnisdatei gespeichert, die Daten in einem zusammenhängenden Bereich des EEPROMs. Datei- und Verzeichnisnamen sind 16 Bits lang (vergleiche Abbildung 2.4).

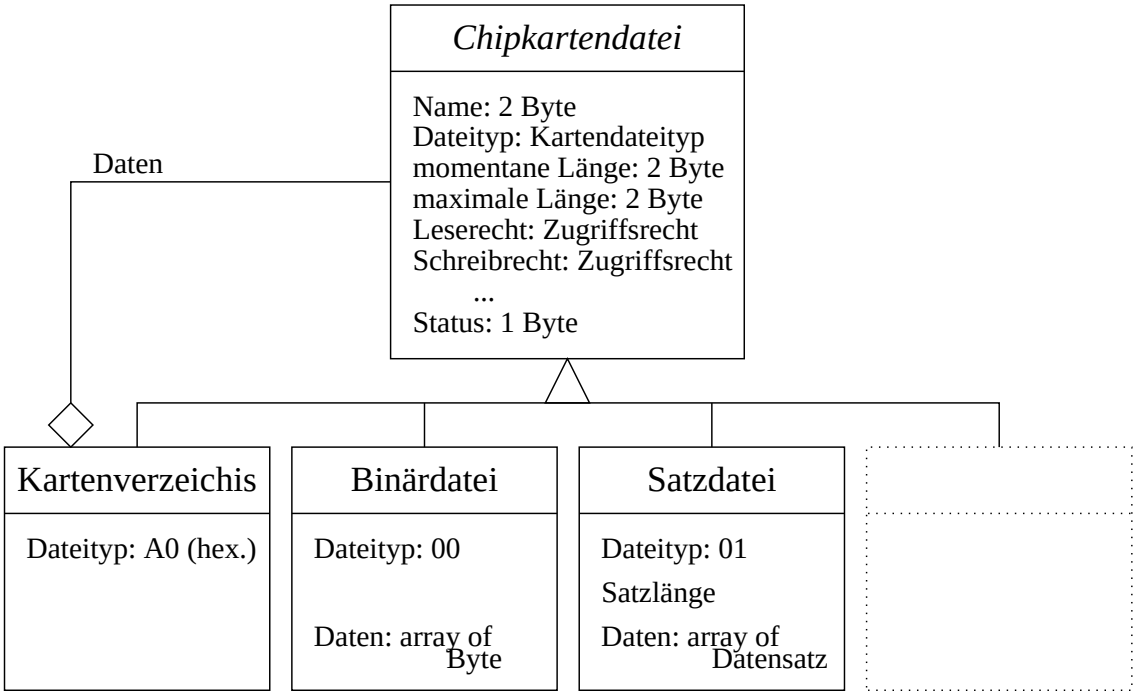

Abbildung 2.4: Dateitypen

Es ist möglich verschiedene Dateitypen zu erzeugen:

- Binäre Dateien (*transparent files*), in denen jedes Byte einzeln angesprochen werden kann.
- Satzorientierte Dateien:
  - Dateien mit fester Satzlänge (*linear fixed*), die Datensätze mit fester Länge enthalten.
  - Dateien mit zyklischer Struktur und fester Satzlänge (*cyclic*), die Datensätze mit fester Länge enthaltenen. Die Adressierung der Datensätze erfolgt relativ zum zuletzt zugegriffenen Datensatz. Bei Erreichen des Dateiendes wird an den Anfang gesprungen.
  - Dateien mit variabler Satzlänge (*linear variable*), die Datensätze werden mit der tatsächlichen Satzlänge gespeichert

- Ausführbare Dateien (*executable*) sind Binärdateien, die von der Karte ausführbaren Code enthalten.

Es kann immer nur eine Datei zur Zeit vom COS geöffnet werden. Auf dieser Datei können verschiedene Operationen ausgeführt werden.

Dateien erzeugen: Um eine Datei zu erzeugen, muß Speicherplatz im Datenbereich der Karte verfügbar und Platz für einen Verzeichniseintrag im Verzeichnis vorhanden sein. Der Dateiname, die maximale Länge der Datei, die Zugriffsrechte und die Startadresse werden in das EEPROM der Karte geschrieben. Das interne Format dieser Daten ist nicht durch die Norm vorgeschrieben und wird von den Herstellern bestimmt. Eine Datei ist benutzbar, wenn ein korrekter Dateikopf im Verzeichnis eingetragen worden ist.

Dateien schreiben: Es wird unterschieden zwischen:

- „Daten ans Dateiende anhängen“(*create*) und
- „Geschriebene Daten ändern“(*update*)

Für beide Befehle werden Kommandos benutzt, die mit unterschiedlichen Zugriffsrechten geschützt werden (z.B. kann das Anhängen von Bytes an eine Datei nur nach vorheriger Eingabe eines Zugriffscodes ermöglicht werden).
Im ISO-Protokoll können maximal $2^8 - 1 = 255$ Bytes mit einem Befehl geschrieben werden, da der Längenparameter eines Befehls 8 Bits lang ist. In vielen Karten ist es nicht möglich, die vollen 255 Bytes zu schreiben, da die Karte diese im RAM zwischenspeichern muß, bevor sie ins (langsame) EEPROM geschrieben werden können. Die genauen Daten finden sich in den Handbüchern der Karten.
Beim Ändern von Daten kann indiziert zugegriffen oder von einem internen Dateizeiger aus geschrieben werden. Dieser Zeiger wird nach der Operation auf das Byte gesetzt, das dem zuletzt geschriebenen Byte folgt.

Dateien lesen: Der Lesezugriff kann durch einen Zugriffscode gesperrt werden und es können maximal 255 Bytes auf einmal gelesen werden (siehe Dateien schreiben).
Ein indiziertes Lesen und ein Lesen vom momentanen Dateizeiger an sind möglich. Der Zeiger wird, wie beim Schreiben, nach der Operation auf das Nachfolgebyte des zuletzt gelesenen Bytes gesetzt.

Zugriffsrechte: Ein wesentlicher Unterschied zu Dateisystemen auf Disketten oder Festplatten ist, daß das Kartendateisystem über einen aktiven Zugriffsschutz verfügt. Dateien können nur bearbeitet werden, wenn bestimmte Zugriffsrechte vorhanden sind. Zugriffsschutz kann für jede Dateioperation einzeln vergeben werden. Schreiben, Lesen und Anhängen von Bytes sind einzeln schützbar (siehe Abbildung 2.5 auf der nächsten Seite).
Die verwendbaren Attribute für den Zugriff sind: Immer, Nie und „nur nach Bekanntgabe eines Zugriffscodes“. Verfügt die Karte über mehrere Zugriffscodes, kann festgelegt werden, welcher für bestimmte Operationen erforderlich ist. Tabelle 2.3 auf der nächsten Seite zeigt beispielhaft Zugriffsrechte für eine Datei: Das Betriebssystem der Karte kann Dateien mit verschiedenen Zugriffcodes (PINs) schützen.

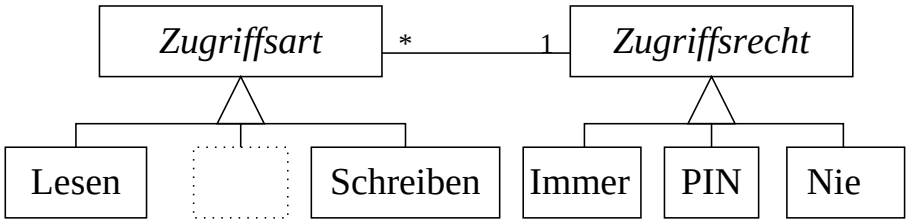

Abbildung 2.5: Zugriffsrechte für eine konkrete Dateioperation

Zugriff	Lesen	Schreiben	Anhängen	Sperren	Entsperren
Bedingung	Immer	PIN 1	PIN 1	Immer	PIN 2

Tabelle 2.3: Beispiel für Zugriffsrechte einer Datei

Wird ein Zugriffscode mehrfach falsch eingegeben, sperrt die Karte weiteren Zugriff auf mit diesem Code geschützte Daten. Nur durch Übermittlung eines Entsperrungscodes kann ein gesperrter Sicherheitscode wieder entsperrt werden (vergleiche Zustandsdiagramm 2.6). Das

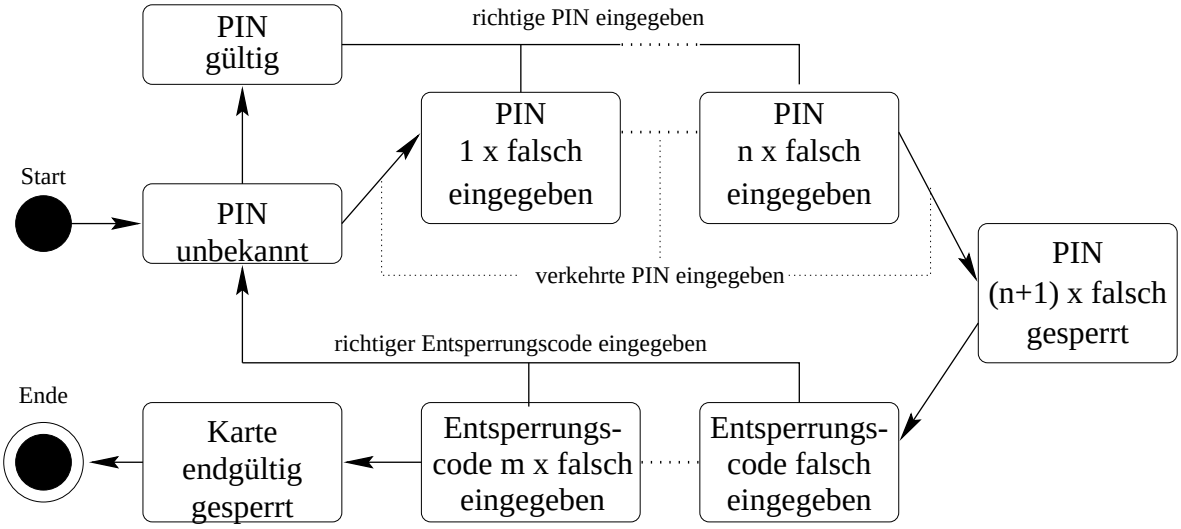

Abbildung 2.6: Zustände der Zugriffscodes

Diagramm zeigt eines der beiden gebräuchlichen Verfahren. Der gesperrte Zugriffscode kann hier durch einen Entsperrungscode entsperrt werden. Das zweite Verfahren erlaubt eine Entsperrung des Zugriffscodes nur, wenn ein entsprechender Entsperrungscode zusammen mit dem gültigen Zugriffscode übermittelt wird.

Das zweite Verfahren eignet sich für fest vom Herausgeber der Karte zugewiesene Zugriffscodes, die während der gesamten Benutzungszeit der Karte gleich bleiben. Das erste kann auch eingesetzt werden, wenn der Benutzer den Code frei wählen und verändern kann.

2.1.2 Verwendete Prozessoren

Die heute in Chipkarten verwendeten Prozessoren sind 8-Bit-Single-Chip Mikrocontroller mit integriertem RAM, ROM und EEPROM. Um diese Controller in Betrieb zu nehmen, werden nur Versorgungsspannung und Takt angelegt. Die geringen Herstellungskosten machen diese Prozessoren für die in großen Stückzahlen hergestellten Chipkarten geeignet. Auf externe Bausteine kann verzichtet werden, damit werden Produktionskosten eingespart.

Häufig werden Derivate des Intel 8051 Mikrocontrollers eingesetzt, die für Chipkartenanwendungen optimiert sind. Diese Controller sind sehr gut getestet. Ein Angriff auf die Sicherheit der Kartendaten kann sich nicht einen Fehler im Mikrocode des Prozessors zunutze machen.

Neuere Entwicklungen sind 16-Bit-Kartenprozessoren, die sich jedoch noch nicht durchgesetzt haben. 32-Bit-Kartenprozessoren werden speziell für die Verwendung in Chipkarten entwickelt (Multi Application Secure SmartCard, MASSC Projekt [DeL 97]) und sind voraussichtlich Mitte 1999 verfügbar.

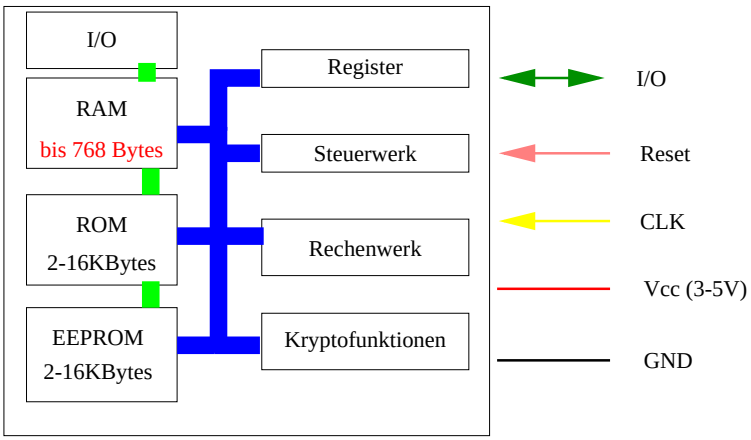

Abbildung 2.7: Funktionsgruppen eines Chipkartencontrollers

Nur die Kontakte des Prozessors, die die ISO-Norm für Chipkarten festlegt, sind zugänglich: I/O, Takt, Reset, Versorgungsspannung und Masse. Adreß- und Datenleitungen sind nicht herausgeführt. Die Prozessoren haben bei 5 V eine Stromaufnahme von 5 bis 10 mA, 3 V Versionen entsprechend weniger. 3-Volt-Prozessoren werden aufgrund des geringeren Stromverbrauches vor allem in mobilen Anwendungen (Mobiltelefone) verwendet.

Zum Schreiben eines EEPROMs sind 12 V erforderlich. Da die Karte nur eine Spannungsversorgung von 5 V hat, verfügt der Prozessor über eine Ladepumpe, die diese Spannung aufbaut. Es dauert bei heutigen Mikrocontrollern weniger als 10 ms, bis diese Spannung aufgebaut ist und der Schreibvorgang einer EEPROM-Zelle durchgeführt werden kann.

Die Kommunikation mit Schreiblesegeräten für Chipkarten wird von der Hardware des Mikrocontrollers unterstützt. Sie implementiert die in Teil 3 der ISO-Norm 7816 spezifizierte bidirektionale I/O-Schnittstelle.

Das Betriebssystem der Chipkarte befindet sich im ROM des Prozessors. Es wird automatisch ausgeführt, wenn Versorgungsspannung angelegt wird. Dieses Verhalten wird *power-on-reset* genannt. Hierüber wird sichergestellt, daß der Controller sich direkt nach dem Anlegen von Versorgungsspannung in einem definierten Zustand befindet.

Aus Sicherheitsgründen wird oft ein Teil des COS im EEPROM abgelegt, da hiermit ein Auslesen der gespeicherten Daten mit chemo-elektrischen Verfahren erschwert wird. Das EEPROM des Controllers befindet sich bei einigen Controllern aus Sicherheitsgründen in tiefergelegenen Schichten der Siliziumstruktur.
Ein 8-Bit-Rechenwerk führt die im Betriebssystem nötigen Berechnungen durch; für die Ausführung kryptographischer Algorithmen existiert häufig ein kryptographischer Koprozessor, der die Ausführung dieser Operationen beschleunigt.

2.1.3 Das Kartenbetriebssystem (*Card Operating System*)

Da die Kartenbetriebssysteme herstellerspezifisch sind und keine Quellen dazu vorliegen, können in diesem Abschnitt nur allgemeine Konzepte erläutert werden, ohne auf Implementierungsdetails einzugehen. Eine Veröffentlichung der Quellen der Betriebssysteme kommt für die Hersteller nicht nur aus Gründen des Kopierschutzes, sondern vor allem aus sicherheitstechnischen Gründen nicht in Frage: Es wäre möglich, Schwachstellen bzw. Fehler zu finden und hierdurch die Sicherheit der Benutzerdaten zu kompromittieren.
Einen allgemeinen Überblick über Funktionsgruppen in Kartenbetriebssystemen stellt das Schema in Abbildung 2.8 (nach [Wolfgang Rankl 96]) dar.

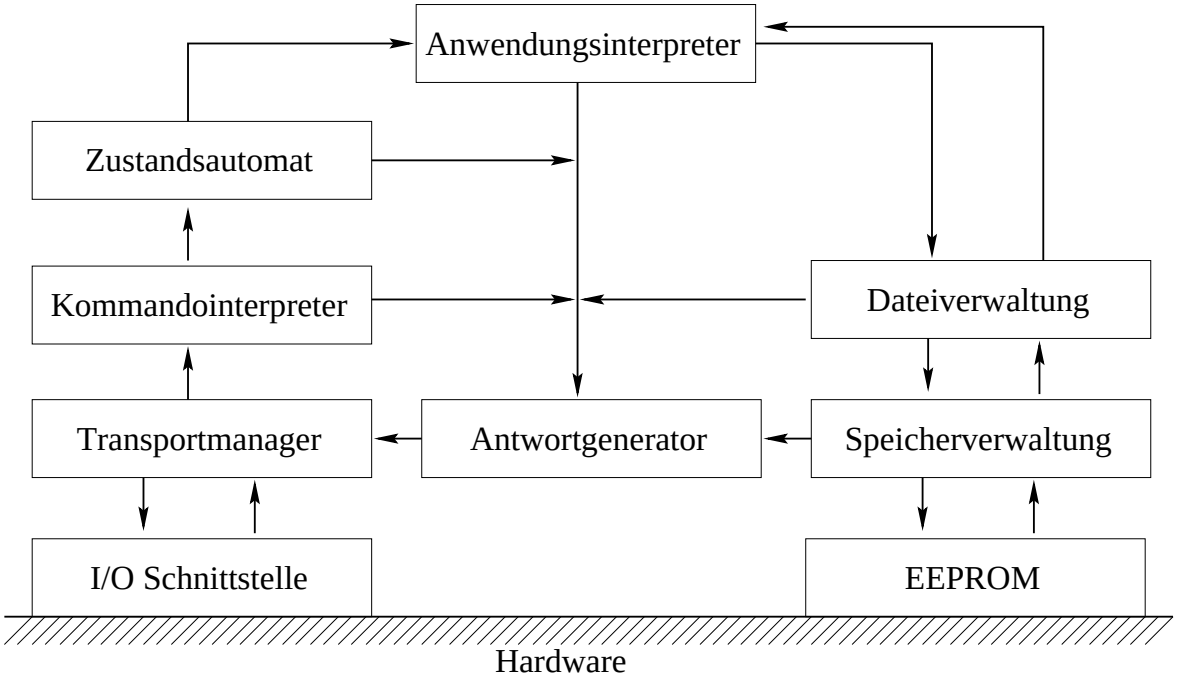

Abbildung 2.8: Befehlsabarbeitung im Kartenbetriebssystem

Anwendungsbefehle werden über die I/O Schnittstelle an den Transportmanager übergeben, der überprüft, ob die Daten im korrekten Format und mit richtiger Prüfsumme übertragen worden sind. Der Kommandointerpreter dekodiert den Befehl. Wenn der Zustand des Zustandsautomaten die Befehlsausführung erlaubt, wird der Befehl vom Anwendungsinterpreter ausgeführt. Zugriff auf Daten im EEPROM kann nur durch das Dateisystem und eine Speicherverwaltung durchgeführt werden.

In jedem der Module ist ein Programmteil enthalten, der überprüft, ob der im Moment ausgeführte Befehl mit dem Sicherheitssystem kollidiert. In diesem Fall wird der Antwortgenerator aufgerufen, der eine Fehlermeldung generiert, ohne daß weitere Teile des Betriebssystems aktiviert werden. Der Antwortgenerator gibt Meldungen der Karte im Format der ISO-Norm aus.
Daten bzw. Dateien im EEPROM sind oft nur nach Bekanntgabe eines Sicherheitscodes zugreifbar. Der gesamte EEPROM-Zugriff ist durch einen Zustandsautomaten geschützt, der den Status der Sicherheitscodes überwacht. Datei- und Speicherverwaltung verhindern einen direkten Zugriff auf Daten des EEPROMs.
Das *card operating system* hat alle Merkmale eines Betriebssystems [Tanenbaum 92]:

- einen Programminterpreter, der Befehle im Stapelbetrieb abarbeitet (und den *scheduler* darstellt),
- eine Speicherverwaltung für die im EEPROM abgelegten Daten,
- eine Betriebsmittelverwaltung, die im Zustandsautomaten Zugriffe auf Kartenfunktionen einschränken kann,
- ein Dateisystem, das es erlaubt, Dateien im EEPROM zu verwalten.

Kartenbetriebssysteme werden für hohe Sicherheitsanforderungen entworfen. Der Schutz vor unberechtigtem Zugriff auf Daten in RAM, ROM und EEPROM ist daher direkt im Betriebssystem verankert. Die Speicherverwaltung erkennt, wenn eine Anwendung auf geschützte Daten zugreift, und generiert eine Fehlermeldung. Das COS befindet sich im ROM der Karte; bei einigen Karten können vor der Auslieferung der Karte Betriebssystemergänzungen im EEPROM abgelegt werden. Die Funktionen des Kartenbetriebssystems werden direkt aus dem nichtflüchtigen Speicher ausgeführt und nicht vorher ins RAM kopiert.
Die Betriebsmittel von Chipkartenprozessoren sind beschränkt. Das RAM erlaubt aufgrund seiner Größe nicht die Ausführung von geladenen Programmen. Karten, die Programme ausführen können, speichern die Programme im EEPROM und führen sie aus dem EEPROM heraus aus. Das RAM wird nur zur Speicherung lokaler Variablen und Stapel genutzt.

2.1.4 Programmausführung auf Chipkarten

Chipkarten, die Programme ausführen können, sind seit 1996 auf dem Markt. Neben Systemen, die herstellerspezifische Programmiersprachen [IBM 98; Dal 97] verwenden, existieren Karten, die JAVA-Bytecode ausführen können.
Die JAVA-Karte enthält im COS einen JAVA-Bytecodeinterpreter, der spezielle JAVA-Programme, *Cardlets* genannt, ausführt. Der Standard Javacard 2.0 [Sun 98b; Sun 98a; Matthias Kaiserswerth 98] spezifiziert die Untermenge von JAVA-Befehlen, die der Bytecodeinterpreter auf der Karte interpretieren kann. Wie ein „normaler“Computer kann eine diesem Standard entsprechende Chipkarte Benutzerprogramme ausführen. Anstelle der unflexiblen statischen Personalisierung (vergleiche Abschnitt 2.1.5) vor Übergabe der Karte an den Endbenutzer, kann die Applikation auch später noch verändert werden. Risiken liegen in fehlerhaften Anwendungen, die die Datensicherheit gefährden können. Der JAVA-Bytecodeinterpreter ist aus diesem Grund auf die Anforderungen von Chipkarten zugeschnitten und verfügt über spezielle Speicherschutzmechanismen.

Minimalanforderungen an eine solche Chipkarte sind: 256 Bytes RAM, 8 KBytes EEPROM und 16 KBytes ROM. Bisher verarbeitet die Javamaschine der Karte nur `Integer` und `String` Datentypen sowie Klassen, die in ihrer Größe von RAM und EEPROM beschränkt sind. Es existiert kein *package-loader* und ist nicht möglich *threads* auszuführen. Für leistungsfähigere Kartenprozessoren ist eine Weiterentwicklung des Javacard-Standards zu erwarten.
Um Programme auf der Karte mit Benutzern kommunizieren zu lassen, muß das JAVA-Kartenprogramm Daten über das Schreiblesegerät austauschen. Die auf der Chipkarte ausgeführten Programme müssen daher das ISO-Protokoll zum Datenaustausch mit dem Schreiblesegerät verwenden. Auf diese Weise können Kartenprogramme in allen Kartenschreiblesegeräten, die das ISO-Protokoll für Chipkarten benutzen, verwendet werden. Der verwendete Bytecode-

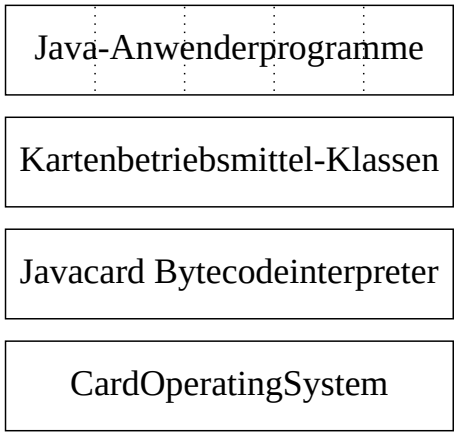

Abbildung 2.9: Integration des JAVA-Bytecodeinterpreters und der Bibliotheken ins COS

interpreter wird vom COS durch einen speziellen Befehl aufgerufen und führt daraufhin das Kartenprogramm in der momentan ausgewählten Datei aus. Der Bytecodeinterpreter hat Zugriff auf das Betriebssystem der Karte und kann ebenfalls auf Kartendateien zugreifen. Um auf Kartenbetriebsmittel zuzugreifen, steht eine Bibliothek zur Verfügung, die auf der Karte ausgeführte Anwendungsprogramme nutzen können. Die Klassen dieser Bibliothek ermöglichen Zugriffe auf Funktionen des Chipkartenbetriebssystems. Das Dateisystem und die kryptographischen Funktionen können auf diese Weise in Chipkartenprogrammen verwendet werden.

2.1.5 Lebenszyklen von Chipkarten

Da Chipkarten in sicherheitssensitiven Anwendungen (Kreditwesen, Krankenversicherung) eingesetzt werden, wird der gesamte Produktionsprozeß von der Herstellung des Chips bis zur Ausgabe der Karte an den Benutzer so durchgeführt, daß es zu keinem Zeitpunkt möglich ist, illegal Daten in den Speicher des Prozessors zu schreiben. Veränderung von Daten im Speicher des Prozessors ist nur durch Übermittlung eines für jede Karte individuellen Schlüssels möglich, der den Zugriff auf den Speicher der Karte freischaltet. Die Chipkarte durchläuft mehrere Phasen, die mit unterschiedlichen Schlüsseln geschützt werden. Ist eine Phase einmal abgeschlossen, so kann nicht mehr in sie zurückgekehrt werden. Es werden fünf Phasen für eine Karte unterschieden (vergleiche Abbildung 2.10):

1. Herstellung (*fabrication*)

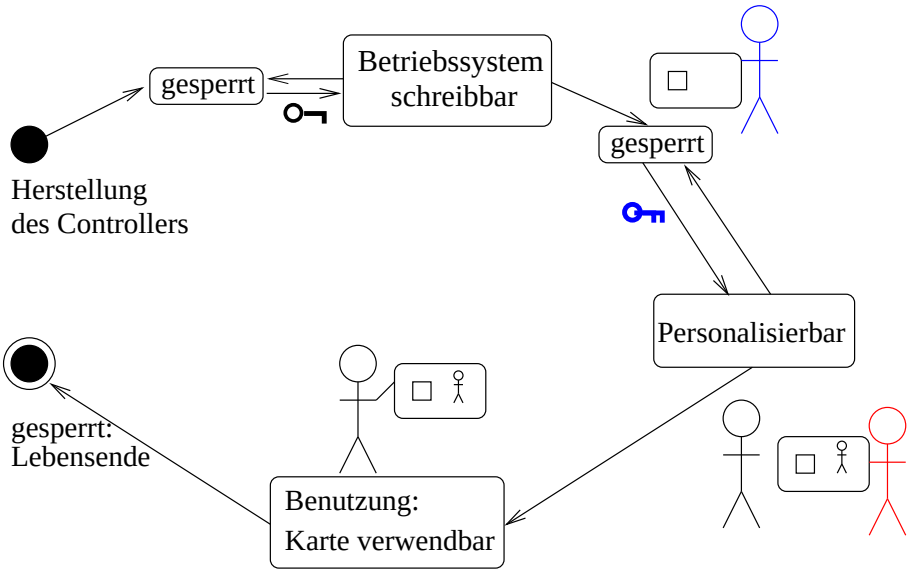

Abbildung 2.10: Schutz der Karten durch Schlüssel

2. Einbau in die Karte (*pre-personalisation*)
3. Personalisierung (*personalisation*)
4. Benutzung (*utilisation*)
5. Lebensende (*end-of-life*)

Die folgenden Abschnitte erläutern kurz die einzelnen Herstellungsphasen.

Herstellung: Der Prozessor wird hergestellt und getestet. Die Herstellungsdaten werden geschrieben. Weiterer Zugriff auf ROM und EEPROM wird durch den Herstellungsschlüssel beschränkt.

Schreiben des Betriebssystems: Der Prozessor wird in eine Kunststoffkarte eingebaut, die Verzeichnisstruktur im EEPROM wird teilweise angelegt und das Betriebssystem wird geschrieben. Der Herstellungsschlüssel wird durch einen Personalisierungsschlüssel ersetzt und eine Sperre geschrieben, die verhindert, in diese Phase zurückzukehren. Das ROM des Controllers ist mit Abschluß dieser Phase nicht mehr schreibbar.

Personalisierung: Die Dateisystemstruktur wird vervollständigt. Die Karte wird mit den Daten des Benutzers der Karte und Anwendungsdaten beschrieben. Um diese Phase abzuschließen, wird der Personalisierungsschlüssel durch einen Kartenbefehl irreversibel gesperrt.

Benutzung: Auf der Karte können nur die bei der Erzeugung der Dateisystemstruktur festgelegten Dateien und Verzeichnisse verändert werden. Das Benutzerkontrollsystem mit den Zugriffscodes ist aktiv. Der Endbenutzer verwendet die Karte. Dies ist „normale“ Phase, in der sich die Karte den größten Teil ihrer Lebenszeit befindet.

Lebensende: Ein weiterer Zugriff auf Kartendaten ist nicht mehr oder nur noch lesend möglich: Die Karte ist mechanisch oder elektrisch verbraucht. Elektrischer Verbrauch heißt, daß das EEPROM nicht mehr in der Lage ist, Daten permanent zu speichern; die Karte detektiert dies und sperrt sich.
Die Karte geht ebenfalls in diesen Zustand über, wenn der Entsperrungscode für Zugriffscodes gesperrt oder der Hauptverzeichniseintrag ungültig geworden ist.

2.1.6 Kryptographische Algorithmen

Um Daten geheimzuhalten, können sie verschlüsselt werden. Viele Chipkarten verfügen über Funktionen, zur Verschlüsselung von Datenblöcken. Die verwendeten Algorithmen werden kryptographische Algorithmen genannt. Kryptographische Algorithmen sind so aufgebaut, daß die Geheimhaltung einer verschlüsselten Nachricht allein auf der Geheimhaltung des Schlüssels beruht. Der verwendete Algorithmus kann veröffentlicht werden.
Es wird zwischen zwei verschiedenen Typen von kryptographischen Verfahren unterschieden: Symmetrischen Verfahren und asymmetrischen Verfahren.
Zwei Arten von Angriffen auf kryptographische Verfahren sind möglich:

- **Analytische Verfahren** machen sich Eigenschaften des Algorithmus zunutze, um auf den Klartext rückschließen zu können.
- **Systematische Verfahren** wenden sich gegen den Schlüssel. Alle möglichen Schlüsselwerte werden zur Entschlüsselung ausprobiert, um den Klartext zu erhalten.

Analytische Verfahren sollen durch den Entwurf des Algorithmus ausgeschlossen werden. Die Sicherheit eines kryptographischen Verfahrens bestimmt sich aus der Rechenzeit, die erforderlich ist, alle möglichen Schlüsselwerte auszuprobieren, um den Schlüssel zu ermitteln. Die Zahl der möglichen verschiedenen Schlüsselwerte bestimmt die Mächtigkeit des Schlüsselraumes. Im Durchschnitt muß der halbe Schlüsselraum durchsucht werden, bis der richtige Schlüssel zum Entschlüsseln einer Nachricht gefunden wird. Die Sicherheit der verschlüsselten Daten steigt mit der Mächtigkeit des Schlüsselraumes des verwendeten Algorithmus. Algorithmen mit unterschiedlich mächtigen Schlüsselräumen benötigen daher unterschiedlich lange Schlüssel, um gleiche Sicherheit zu gewährleisten.

Symmetrische kryptographische Verfahren

Symmetrische kryptographische Verfahren sind Verfahren, die zum Ver- und Entschlüsseln den gleichen Schlüssel verwenden.
Klartext wird mit Hilfe eines Schlüssels in verschlüsselten Text transformiert und kann mit dem gleichen Schlüssel wieder in Klartext transformiert werden (vergleiche Abbildung 2.11 auf der nächsten Seite).
Um eine geheime Nachricht zu verschicken, müssen Sender und Empfänger über den selben Schlüssel verfügen: Der initiale Schlüsselaustausch ist daher bei ausschließlicher Verwendung von symmetrischen kryptographischen Verfahren immer unsicher, da dieser Schlüssel unverschlüsselt übertragen werden muß. Es werden deshalb zum initialen Schlüsseltausch asymmetrische kryptographische Verfahren verwendet.
Verwendete Algorithmen sind *data encryption standard* (DES) Algorithmus mit 56 Bits, Triple-DES mit 128 Bits und *international data encryption algorithm* (IDEA) mit 128 Bits Schlüssellänge. Die Mächtigkeit des Schlüsselraumes bei den Verfahren entspricht der Mächtigkeit des

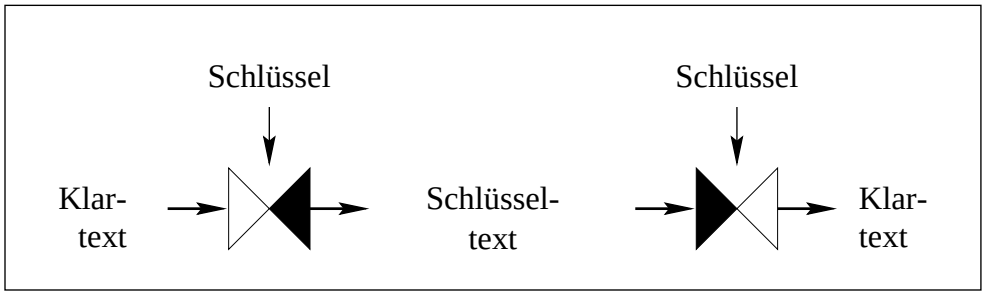

Abbildung 2.11: Verwendung eines symmetrischen kryptographischen Algorithmus

Raumes der dualen Zahlen mit entsprechender Stellenzahl. Ein zusätzliches Bit an Schlüssellänge **verdoppelt** die Rechenzeit, die benötigt wird, um alle möglichen Schlüssel auszuprobieren.

Asymmetrische kryptographische Verfahren

Asymmetrische kryptographische Verfahren sind Verfahren, die zum Entschlüsseln einen anderen Schlüssel verwenden als zum Verschlüsseln. Es sind zwei Verfahren gebräuchlich: Der RSA-Algorithmus und der Diffie-Hellman Algorithmus. Auf Chipkarten, die asymmetrische kryptographische Verfahren beherrschen, hat sich der RSA-Algorithmus durchgesetzt. Dieser Algorithmus wird im Rahmen dieser Arbeit verwendet und im Detail beschrieben.

RSA-Algorithmus Die Abkürzung RSA steht für die Initialen seiner Entwickler Ronald Rivest, Adi Shamir und Leonard Adleman. Ist einer der beiden Schlüssel bekannt, so kann durch diesen nicht auf den anderen zurückgeschlossen werden. Es wird hier unterschieden zwischen öffentlichen Schlüssel (*public key*) und geheimen Schlüssel (*private key*). Das RSA-Verfahren ist blockorientiert Es werden Datenblöcke fester Länge verschlüsselt. Kürzere Blöcke müssen bis zur Blocklänge mit Daten aufgefüllt werden; längere Datenblöcke müssen in mehrere Blöcke unterteilt werden.
Der Algorithmus basiert auf dem Prinzip, daß es sehr einfach ist, zwei (sei es auch große) Zahlen zu multiplizieren, aber sehr zeitaufwendig, ihre Primfaktorzerlegung zu berechnen (mathematische Beschreibung siehe Anhang C auf Seite 80). RSA-Algorithmen werden von Chipkarten mit bis zu 1024 Bits Blocklänge berechnet.
Die Mächtigkeit des Schlüsselraumes entspricht der Zahl aller Primzahlen mit entsprechender Stellenzahl, sie ist daher deutlich kleiner als die Mächtigkeit der Schlüsselräume bei symmetrischen Verfahren gleicher Schlüssellänge.

Verwendung des RSA-Algorithmus Der RSA-Algorithmus kann zum Ver- und Entschlüsseln von Daten verwendet werden. Beim Verschlüsseln wird mit dem öffentlichen Schlüssel ein Klartextblock verschlüsselt, der nur mit Hilfe des geheimen Schlüssels entschlüsselt werden kann (vergleiche Abbildung 2.12 auf der nächsten Seite).
Der zweite Einsatzbereich von asymmetrischen kryptographischen Verfahren ist die digitale Signatur. Zur Berechnung einer digitalen Signatur wird eine Prüfsumme (*hash*-Wert) für die zu signierenden Daten berechnet. Das heißt, es wird aus einem Klartextblock größerer Länge

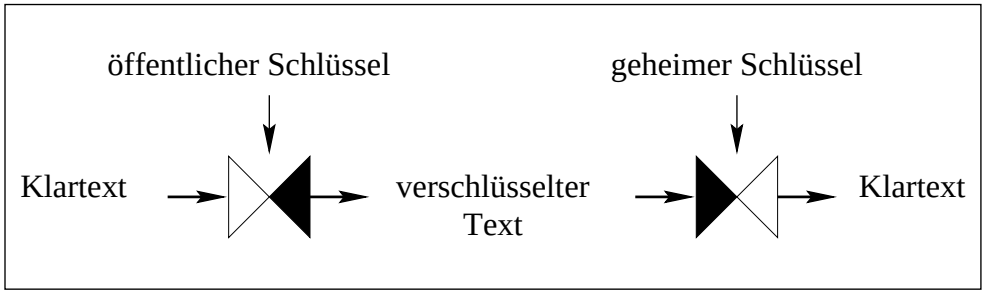

Abbildung 2.12: Verschlüsselung von Klartext

durch ein Einwegverfahren ein Wert berechnet, in den alle Bits des Klartextes eingehen. Dieser Block hat eine Länge, die der Blocklänge des Verschlüsselungsalgorithmus entspricht (siehe Abbildung 2.13). Die Prüfsumme wird dann mit dem geheimen Schlüssel verschlüsselt.

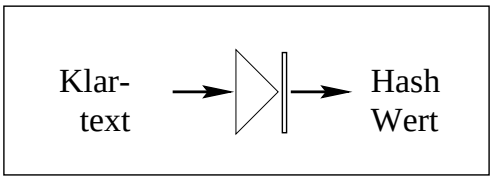

Abbildung 2.13: Berechnung einer kryptographischen Prüfsumme

Soll überprüft werden, ob eine empfangene signierte Nachricht authentisch ist, wird die verschlüsselte Prüfsumme mit dem öffentlichen Schlüssel entschlüsselt und der berechnete Wert mit dem über die geschickten Daten berechneten verglichen (vergleiche Abbildung 2.14).

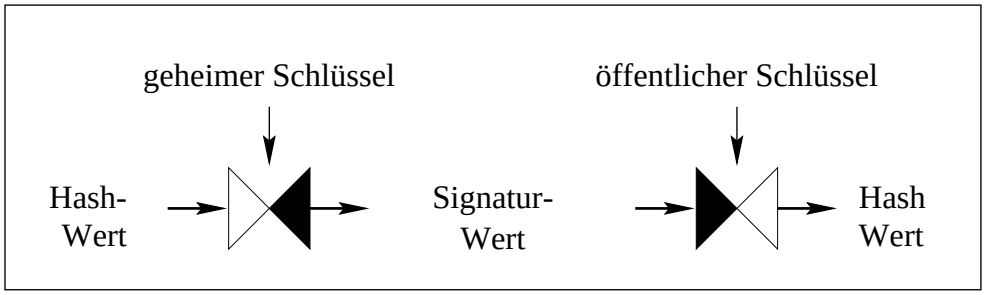

Abbildung 2.14: Berechnung und Verifikation einer digitalen Signatur

Es ist nur mit dem geheimen Schlüssel möglich, eine korrekte Signatur zu generieren, obwohl der Verschlüsselungsalgorithmus, der öffentliche Schlüssel und die Funktion zur Berechnung der Prüfsumme bekannt sind. Das beschriebene Verfahren wird dazu eingesetzt, die Authentizität von Daten zu überprüfen.

2.1.7 Klassifikation von Chipkarten nach ihrem Sicherheitssystem

Chipkarten werden für einen bestimmten Einsatzbereich entwickelt: Es werden sehr spezialisierte Karten für einen einzigen Aufgabenbereich produziert, aber es gibt auch Karten mit erweiterbarer Struktur, die zur gleichen Zeit Daten mehrerer Anwendungen voreinander geschützt enthalten können.
Die Karten können nach dem für die Anwendung erforderlichen Sicherheitsniveau klassifiziert werden. Vier verschiedene Typen von Chipkarten sind auf diese Art unterscheidbar:

- Karten ohne Sicherheitsmechanismen
- Karten für geringe Sicherheitsanforderungen
- Karten für mittlere Sicherheitsanforderungen
- Karten für hohe Sicherheitsanforderungen

Keine Sicherheitsmechanismen

Speicherkarten sind die einfachste Form der Chipkarte: Das COS erlaubt den Zugriff auf den Speicher der Karte, ohne daß dieses mit Zugriffsbedingungen verknüpft werden kann. Ein Dateisystem ist nicht vorhanden. Diese Karten werden in Massenanwendungen verwendet, in denen keine sicherheitssensitiven Daten auf der Karte gespeichert werden. Die in Deutschland verwendete Krankenversicherungskarte ist eine solche Karte: Nur die auf dem Kartenkörper lesbare Information ist auch innerhalb der Karte gespeichert.

Geringe Sicherheitsanforderungen

Chipkarten für geringe Sicherheitsanforderungen erlauben es, Kartendaten durch einen Zugriffscode zu schützen. Ein komplexes Dateisystem fehlt auf diesen Karten.
Diese Karten werden vor allem in Massenanwendungen mit geringem Speicherbedarf verwendet, da sie sehr preiswert sind. Es sind ebenfalls Karten erhältlich, die anstelle der elektrischen Kontakte einen Sender- und Empfänger enthalten, der sie über Funk auslesbar macht.
Ein typisches Beispiel sind Identifikationskarten, die in elektronischen Schließsystemen eingesetzt werden und über wenig Speicher verfügen (ca. 100 Bytes).

Mittlere Sicherheitsanforderungen

Chipkarten für mittlere Sicherheitanforderungen werden für verschiedene Aufgaben entworfen. Um flexibel einsetzbar zu sein, verfügen diese Karten über ein Dateisystem mit Unterverzeichnissen. Unterverzeichnisse enthalten Daten für einzelne Applikationen. Es stehen bis zu 16 KBytes EEPROM zur Verfügung. Dateien und Verzeichnisse können mit verschiedenen Zugriffscodes geschützt werden und es ist möglich, neue Dateien und Verzeichnisse in der Benutzungsphase anzulegen. Das Dateisystem implementiert den vollen Funktionsumfang der ISO 7816-4 Norm.
Durch die Zugriffsrechte wird die Verwendung der einzelnen Funktionen so eingeschränkt, daß z.B. Veränderungen des Dateisystems (Erzeugen/Löschen von Dateien) über einen anderen Zugriffscode geschützt werden als das Schreiben in Dateien.

Karten für mittlere Sicherheitsanforderungen sind dazu in der Lage, einen kryptographischen Algorithmus auszuführen (meist DES). Die verwendbaren Schlüssellängen sind beschränkt. Der Algorithmus dient der Überprüfung, ob die Karte echt ist und der digitalen Signatur von Daten. Verschlüsselungen von geheimzuhaltenden Daten können nur mit kurzen und deshalb unsicheren Schlüsseln durchgeführt werden.
Die Karten werden von ihrem Benutzer für mehrere Anwendungen verwendet. Ein typisches Beispiel sind Multifunktionskarten, die als elektronische Schlüssel eingesetzt werden, zum Anmelden auf dem Rechner verwendet werden und über die Möglichkeit verfügen, einfache bargeldlose Bezahlungsfunktionen auszuführen.

Hohe Sicherheitsanforderungen

Karten mit kryptographischen Funktionen: Ein oder mehrere kryptographische Algorithmen können ausgeführt werden, um Daten zu verschlüsseln. Die Algorithmen können zur Authentifizierung der Karte, zum Verschlüsseln und zum Signieren von Daten verwendet werden.
Gebräuchlich sind DES-, Triple-DES- und RSA-Algorithmen sowie kryptographische Prüfsummen. Die verwendbaren Schlüssellängen sind zur Verschlüsselung geheimer Daten geeignet.
In sicherheitssensitiven Anwendungen werden normalerweise Chipkarten für einen einzigen Aufgabenbereich entworfen. Hier wird zugunsten von Sicherheit auf eine flexible Struktur verzichtet. Dateien können nicht gelöscht werden. Es ist nicht möglich, neue Dateien zu erzeugen. Die meisten Funktionen auf solchen Karten werden mit Zugriffscodes geschützt.
Speicherkapazität und flexible Dateierzeugung sind weniger ausgeprägt, da die Karte vom Benutzer für eine einzige Anwendung eingesetzt wird.

Bankkarten: Diese Karten werden für Abbuchungs- und Zahlungsvorgänge verwendet und verfügen über ähnliche Funktionalität wie Multifunktionskarten. Es ist während der Benutzung nicht mehr möglich, das Dateisystem zu modifizieren. Die Karten verfügen über mehrere kryptographische Funktionen (DES, RSA) zur Verschlüsselung und Authentifizierung.
Diese Karten sind nur in modifizierter Form für Nicht-Bankkunden erhältlich.

2.1.8 Kartenschreiblesegeräte

Das Kartenschreiblesegerät ermöglicht den Datenaustausch mit der Chipkarte. Schreiblesegeräte verschiedener Art sind verfügbar. Im Rahmen dieser Arbeit werden nur Geräte mit Anschlußmöglichkeit an einen Computer betrachtet; Lösungen mit direkt im Schreiblesegerät programmierten Funktionen werden nicht betrachtet.
Die Aufgaben des Schreiblesegerätes sind, die Karte mit Spannung und Takt zu versorgen, sowie Kommunikationsprotokollblöcke mit der Chipkarte bei geringer Datenrate (9600 bit/s) auszutauschen.
Schreiblesegeräte sind durch die verschiedenen Anschlußmöglichkeiten an den Rechner charakterisiert. Geräte für alle externen Schnittstellen von Computersystemen sind verfügbar:

- Geräte mit serieller Schnittstelle (über eine RS-232-Schnittstelle an den Kontrollrechner anschließbar)
- Geräte mit paralleler Schnittstelle (über den Druckeranschluß anschließbar)

- Einschubkarten für den Bus des PCs (XT-, AT- oder PCI Bus)
- PCMCIA Geräte für PC-Card-Steckplätze
- Geräte für das Diskettenlaufwerk (ein Hardwarezusatz setzt die elektrischen Signale in magnetische Signale um)

Intern nutzen alle diese Geräte das ISO-Protokoll, um Daten mit der Karte auszutauschen. Unterschiedlich ist die Art, wie diese Daten an den Kontrollrechner weitergegeben werden.

2.1.9 DeLaRue DXPLUS Chipkarte

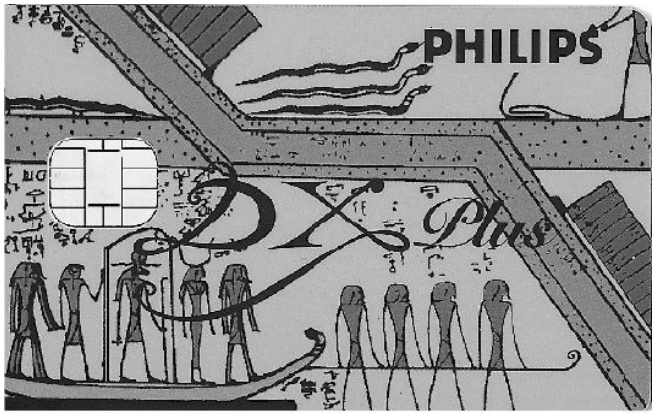

Abbildung 2.15: Die DeLaRue DXPLUS Chipkarte

Die am Arbeitsbereich für Softwaresysteme verwendete Chipkarte (siehe Abbildung 2.15) wird von DeLaRue CardSystems in Frankreich hergestellt. Technische Daten dieser Karte finden sich in Tabelle 2.4 auf der nächsten Seite.
Die Karte erfüllt die ISO-Norm 7816 in den Teilen 1-3 bezüglich physikalischer und elektrischer Daten sowie der verwendeten Übertragungsprotokolle. Fehlererkennung bei der Datenübertragung wird byteweise durchgeführt ($T = 0$ Verfahren, vergleiche Abschnitt 2.1.1 auf Seite 11).

Dateisystem

Das Dateisystem der DXPLUS-Karte ist teilweise ISO-konform. Es wird eine Untermenge der in ISO 7816-4 standardisierten Funktionalität implementiert.
Auf der Karte können nur Binärdateien (*binary transparent files*) erzeugt werden. Alle Dateien werden im Hauptverzeichnis angelegt; es ist nicht möglich, weitere Verzeichnisebenen zu erzeugen. Dateien sind über fünf Befehle ansprechbar:

- `select` wählt eine Datei zur Bearbeitung aus,
- `readBinary` liest Bytes aus einer Datei,
- `updateBinary` verändert geschriebene Bytes,

Mikrocontroller	Philips 83C858
Versorgungsspannung	5 V
RAM Größe	640 Bytes
ROM Größe	6 KBytes
Datenspeicher	EEPROM 8 KBytes
Kryptographischer Algorithmus Blocklänge	RSA bis 1024 Bits
Protokoll	byteweise Fehlererkennung ($T = 0$)
implementierte Norm	ISO 7816-3
Maximale Taktrate	8 MHz
Datenübertragungsgeschwindigkeit	9600 bit/s (3,57 MHz) bis 21000 bit/s (8MHz)

Tabelle 2.4: Technische Daten der DeLaRue DXPLUS Chipkarte

- `createBinary` hängt Bytes an eine Datei an,
- `invalidate` sperrt eine Datei für weiteren Zugriff.

Jede dieser Operationen verfügt über eine eigene Zugriffskontrolle. Vergebbare Zugriffsrechte für die Operationen sind:

- „Nie ausführbar“,
- „Immer ausführbar“und
- „nach Bekanntgabe eines Zugriffcodes ausführbar“.

Verschlüsselungsfunktionen der DXPLUS-Karte

Die DXPLUS-Chipkarte ist dazu in der Lage, Daten mit dem RSA-Algorithmus zu ver- und entschlüsseln. Der zu verwendende Schlüssel muß in einer Datei auf der Chipkarte gespeichert sein. Auf der DXPLUS Chipkarte können Blöcke von bis zu 1024 Bits Länge mit dem RSA Algorithmus ver- bzw. entschlüsselt werden (siehe Anhang C auf Seite 80).
Die Ausführung des Algorithmus kann von der Eingabe eines Zugriffscodes abhängig gemacht werden.

Ausführungszeiten des RSA-Algorithmus

Die Chipkarte kann einen Datenblock von 1024 Bits Länge bei 8 MHz Takt in 300 ms mit dem RSA-Algorithmus ver- oder entschlüsseln; Vergleichswerte zeigt Tabelle 2.5 auf der nächsten Seite.
Der RSA-Algorithmus darf aus den Vereinigten Staaten nur mit Schlüssellängen bis zu 40 Bits exportiert werden. Das Ausführen des Algorithmus kann auf der DXPLUS Chipkarte irreversibel einzuschränkt werden. Hierzu wird eine Sperre in die Karte geschrieben. Der Algorithmus kann auf drei verschiedene Arten gesperrt werden:

1. vollständig abgeschaltet,

Schlüssellänge/Bits	512		768		1024	
Takt/MHz	3,57	8	3,57	8	3,57	8
Zeit/ms	170	75	360	160	650	300

Tabelle 2.5: Ausführungszeiten des RSA-Algorithmus mit dem chinesischen Restklassensatz

2. auf Berechnungen mit dem chinesischen Restklassensatz beschränkt,
3. nur zum Signieren verwendbar.

Benutzung des RSA- Algorithmus auf der DXPLUS Chipkarte

Um die Chipkarte den RSA-Algorithmus ausführen zu lassen, wird ein entsprechender Befehl an die Chipkarte gesendet. Das Resultat wird mit einem anderen Befehl ausgelesen. Die Vorgehensweise ist wie folgt:

1. Auswahl der Schlüsseldatei (öffentlicher oder geheimer Schlüssel) mit `select`,
2. Übermittlung des Zugriffscodes falls erforderlich (`verifyPIN`),
3. Ausführung von `computeRSA`,
4. Abholen des Ergebnisses mit `getResponse`.

2.1.10 Das PHILIPS PE122-210 Chipkartenschreiblesegerät

Das PE122-210 Chipkartenschreiblesegerät von PHILIPS ist an die RS-232-Schnittstelle eines Computers anschließbar. Es unterstützt die beiden in der ISO-Norm standardisierten asynchronen Protokolle ($T = 0$ und $T = 1$), das synchrone I^2C Protokoll und das synchrone ISO-Protokoll. Das Gerät benötigt eine externe 5 V Spannungsversorgung, die auch aus dem Tastaturanschluß eines PCs entnommen werden kann.
Das PE122 taktet Chipkarten nicht mit 3,57 MHz, sondern mit 4 MHz oder 8 MHz. ISO 7816-3 [ISO 89] sieht eine Taktung bis 5 MHz für alle Chipkarten vor, so daß bei 4 MHz alle der Norm entsprechenden Karten betrieben werden können. Datenübertragung zum angeschlossenen PC oder der Workstation ist mit 1200 bis 19200 bit/s möglich.
Mehr als 5 MHz Taktfrequenz werden heute von vielen Chipkarten unterstützt, da die Ausführungszeiten für kryptographische Algorithmen sich durch höhere Taktung proportional verkürzen. Die Datenübertragungsrate wird auf die gleiche Weise erhöht.

2.2 Programmierschnittstellen für Chipkarten in JAVA

Im Rahmen dieser Arbeit wird von der Programmiersprache JAVA aus auf Chipkarten zugegriffen. Dieser Abschnitt beschreibt die Schnittstellen von JAVA zu Chipkarten.

2.2.1 Sprachbeschreibung

JAVA ist eine objektorientierte, plattformunabhängige Programmiersprache, die von der Firma SUN MICROSYSTEMS entwickelt worden ist. In dieser Arbeit wird die Version 1.1.5 des JAVA-DEVELOPMENT-KIT (JDK) verwendet. Merkmale von JAVA werden in den folgenden Abschnitten kurz beschrieben.

Plattformunabhängigkeit Plattformunabhängigkeit wird bei JAVA durch Interpretieren von maschinenunabhängigem Bytecode erreicht. Der Bytecode wird auf der jeweiligen Zielmaschine von einer virtuellen Maschine, dem Bytecodeinterpreter, ausgeführt.
Ein JAVA-Programm wird auf einer beliebigen Maschine in JAVA-Bytecode übersetzt und ist auf allen Maschinen, für die eine Portierung des Interpreters existiert, lauffähig.
Das interne Format aller Datentypen ist festgelegt, z.B. wird der Typ `int` immer mit 32 Bits Genauigkeit und in Zweierkomplement-Darstellung bei negativen Zahlen repräsentiert. Diese Darstellungen werden vom Bytecodeinterpreter in die jeweiligen Darstellungen der einzelnen Maschinen übersetzt.

Objektorientierung JAVA ist vollständig objektorientiert konzipiert, die Syntax ist an C++ angelehnt. Im Vergleich zu C++ werden Mehrfachvererbung und Operatorüberladung nicht unterstützt. Es gibt keine Zeigertypen, alle Objekte werden dynamisch erzeugt. Eine Speicherverwaltung mit Speicherbereinigungsmechanismus (*garbage collection*) übernimmt die Zuteilung von freiem Speicher.

Standardisierte Bibliotheken Eine große Zahl von dynamisch ladbaren Klassenbibliotheken, in JAVA Pakete (`packages`) genannt, macht JAVA zu einer für viele Einsatzbereiche geeigneten Sprache. Pakete, die plattformunabhängige Fenstergrafik, Datenbankzugriff und kryptographische Funktionen zur Verfügung stellen, sind für das JAVA-System verfügbar.

Netzwerkfähigkeit JAVA-Programme können über das Internet geladen und lokal im Webbrowser ausgeführt werden. Zugriff auf das Netzwerk ist bereits in den Standardbibliotheken vorgesehen; es existiert die Möglichkeit, Methoden von Objekten über das Netzwerk aufzurufen (*remote method invocation*, RMI).

Sicherheitskonzept Die Tatsache, daß JAVA-Bytecode über das Netzwerk geladen werden kann, um dann ausgeführt zu werden, erfordert ein Sicherheitskonzept. Der empfangene Bytecode kann vom Interpreter vor der Ausführung verifiziert werden, um zu überprüfen, daß Dateien nicht unerlaubt manipuliert werden.

2.2.2 Chipkartenzugriff mit JAVA

In den beiden folgenden Abschnitten werden zwei Möglichkeiten vorgestellt, Chipkarten mit JAVA anzusprechen. Das Paket `javax.smartcard` enthält nur ein Minimum an Klassen, vor allem zur Verwendung des standardisierten ISO-Protokolles von JAVA aus. Das OPENCARD-Framework ist ein vollständiges objektorientiertes Gerüst, das die Kommunikation zwischen Karte und Schreiblesegerät ermöglicht. Es wird im Rahmen dieser Arbeit verwendet, um mit Chipkarten über ein Schreiblesegerät zu kommunizieren.

Javax.smartcard

Von SUN MICROSYSTEMS werden zusätzlich zu den Basisklassen des JDK Erweiterungen für spezielle Aufgaben zur Verfügung gestellt. Für Chipkarten existiert das Paket `javax.smartcard`, mit dem Chipkarten und Schreiblesegeräte auf Protokollebene angesprochen werden. Aufgabe dieses Paketes ist es, das ISO-Datenaustauschprotokoll der Chipkarte von allen JAVA-Programmen aus auf gleiche Art ansprechen zu können. Klassen, um Kartenschreiblesegeräte anzusprechen, sind ebenfalls vorhanden.

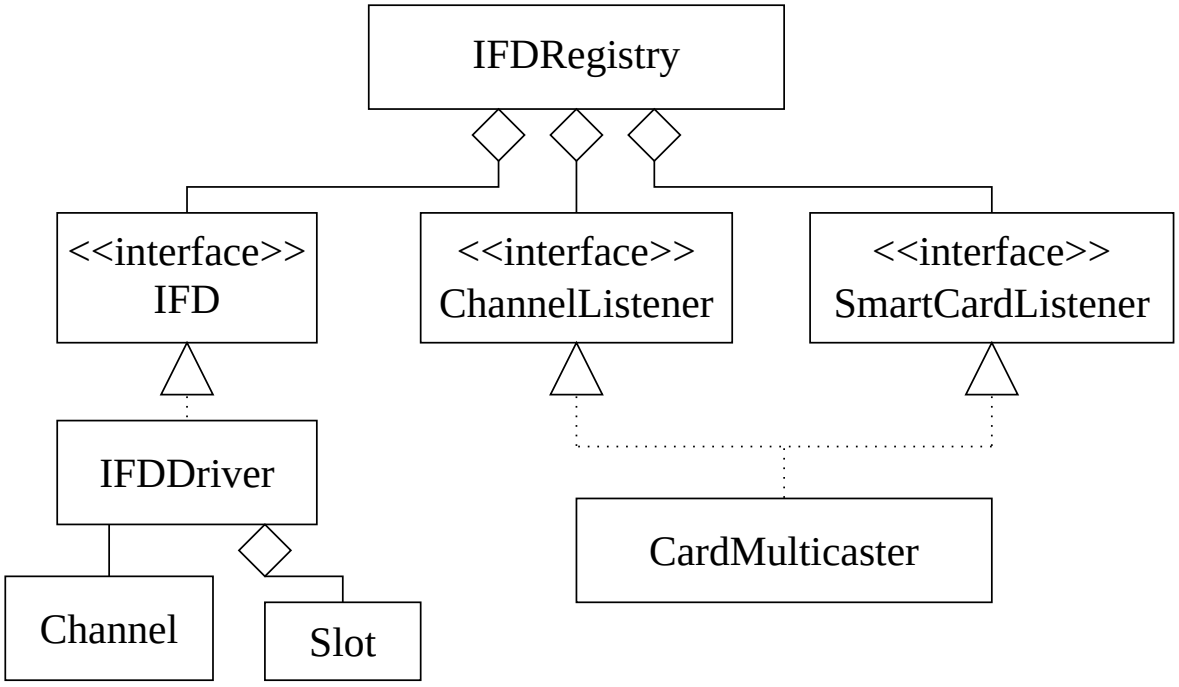

Abbildung 2.16: Klassendiagramm des Paketes `javax.smartcard`

ISO-Protokollblöcke werden in Protokollblock-Objekte gekapselt und können durch einen Treiber für ein Schreiblesegerät an eine Karte geschickt werden. Die Antwort der Karte wird ebenfalls in einem Objekt gekapselt und der Anwendung übermittelt. Die Klassenhierarchie (vergleiche Abbildung 2.16) ist so aufgebaut, daß Kartenschreiblesegeräte (*interface devices*, IFDs) einfach integriert werden können. Treiber für alle Kartenschreiblesegeräte eines Systems können aus einer Registrierung angefordert werden.

Kommunikation mit einem Schreiblesegerät wird über einen Treiber, der Kanäle (`Channel` Objekte) zu physikalischen Karteneinschüben (`Slot` Instanzen) erzeugen kann, betrieben.
Durch eine Ereignisverwaltung (`SmartCardListener`, `ChannelListener`) können Programme benachrichtigt werden, wenn eine Chipkarte in ein Schreiblesegerät eingelegt wird.
Das Paket `javax.smartcard` enthält Klassen, die auf Ebene des ISO-Protokolls den Zugriff auf Chipkarten ermöglichen. Um Treiber für spezielle Chipkarten zu instantiieren, existiert kein standardisierter Mechanismus.

Das OpenCard-Framework

Um die Kommunikation von Chipkarte und Schreiblesegerät auf programmiersprachlicher Ebene zu standardisieren, hat sich eine Arbeitsgruppe verschiedener Chipkartenhersteller, Schreiblesegerätehersteller und Softwarehäuser gebildet, die OpenCard-Gruppe [OpenCard 98].
Es ist ein objektorientiertes Referenzmodell [Herrmann, Husemann 98] entwickelt und in Java implementiert worden, das Zugriff auf Chipkarten und Schreiblesegeräte ermöglicht. Das Modell ist so konzipiert, daß Treiber für neue Karten und Schreiblesegeräte mit geringem Aufwand integriert werden können und die Funktionen einfach benutzbar sind. Die drei Ziele dieses Modells sind:

- Unabhängigkeit vom Schreiblesegerät,
- Unabhängigkeit vom Kartenbetriebssystem und
- Unabhängigkeit vom der Anwendung.

Die Funktionen der Chipkarte werden im OpenCard-Modell als Dienste angesehen, die die Karte gegenüber dem Rechnersystem, das sie anspricht, erbringt. Es wird nicht unterschieden zwischen Diensten der Karte und dem Zugriff auf den Dienst über einen Diensttreiber.
Im folgenden werden die Teile des Systems erläutert, die ein Softwareentwickler verwendet. Die vollständige Dokumentation findet sich in [Ope 98].

Übersicht über die Klassen

Durch die statischen Methoden der Klasse `SmartCard` wird das System initialisiert (Programmbeispiel siehe Anhang A auf Seite 71). Abbildung 2.17 auf der nächsten Seite zeigt eine typischen Ablauf zur Verwendung von Chipkarten. Instanzen der Klasse `SmartCard` haben Zugriff auf in Kartenschreiblesegeräte eingelegte Karten. Objekte dieser Klasse können durch die Ereignisverwaltung benachrichtigt werden, daß eine Karte in ein Gerät eingelegt worden ist. Daraufhin können Diensttreiber für Funktionen der Karte instantiiert werden. Den Zugriff von Diensttreibern auf die Kartenfunktionen regelt ein Arbitrierungsmechanismus, (`CardServiceScheduler`), der verhindert, daß zwei Prozesse zur gleichen Zeit auf eine eingelegte Chipkarte zugreifen.
Der Zugriff von Kartendiensttreibern auf die Funktionen der Chipkarte wird von Instanzen der Klasse *SmartCard* abgewickelt. Diese Klasse verfügt über Methoden, um mit Chipkarten über Kartenschreiblesegeräte zu kommunizieren.
Ein bestimmter Kartentyp wird an der Antwort auf ein Resetsignal erkannt, daraufhin können Diensttreiber für Kartendienste für diese Karte angefordert werden.

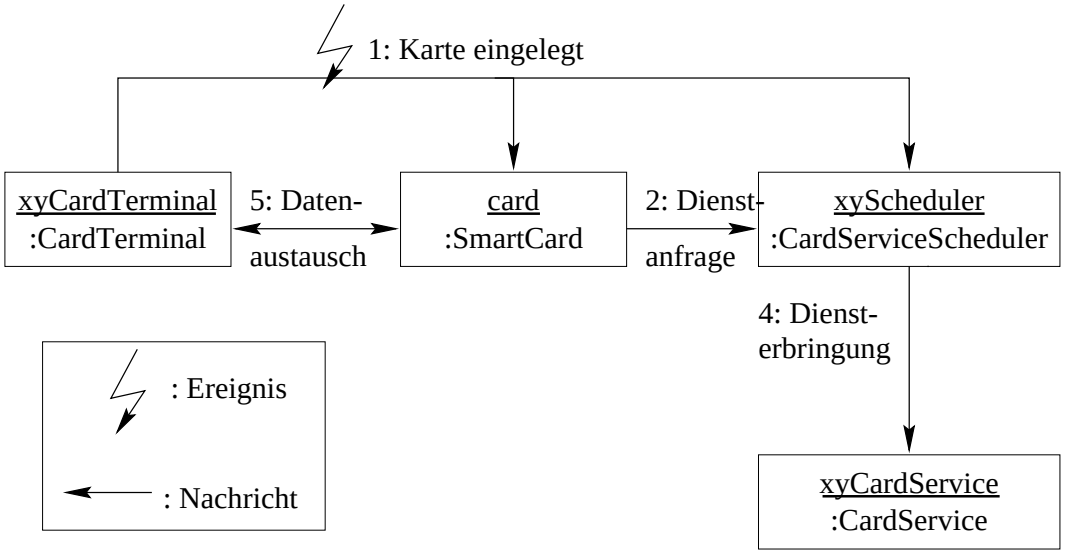

Abbildung 2.17: Übersicht über die Funktionsweise des OPENCARD-Modells

Kartenschreiblesegerät

Das Schreiblesegerät erlaubt den physikalischen Zugriff auf die Chipkarte. Diese Funktionalität wird in der Klasse `CardTerminal` zur Verfügung gestellt (siehe Klassendiagramm in Abbildung 2.18 auf der nächsten Seite). Der Zugriff auf die Karte erfolgt über eine Instanz der Klasse `Slot`, die einen physikalischen Karteneinschub des Gerätes repräsentiert. Zugriffe auf diese Hardware werden von `SlotChannel` Objekten vorgenommen. Hiermit wird sichergestellt, daß nur ein Kanal für jede Karte und jedes Schreiblesegerät zur Zeit geöffnet ist, um Zugriffskonflikte bei Hardwarezugriffen zu verhindern. Durch `SlotChannel` Objekte werden Protokollblöcke an die Karte gesendet und die Antworten empfangen.
Treiber für Kartenschreiblesegeräte werden durch die Verwendung des objektorientierten Entwurfsmusters der abstrakten Fabrik [Gamma et al. 95] instantiiert. Die OPENCARD-Implementierung verwendet anstelle einer abstrakten Basisklasse für Schreiblesegeräte eine Schnittstelle (*interface*) `CardTerminalFactory`, die einer abstrakten Klasse mit ausschließlich abstrakten Methoden entspricht. Konkrete Fabriken für Schreiblesegerättreiber sind Implementierungen dieser Schnittstelle und instantiieren Treiber für Kartenschreiblesegeräte. Die in Abbildung 2.18 auf der nächsten Seite dargestellten Klasse `XYCardTerminalFactory` erzeugt Treiberklassen für Chipkartenschreiblesegeräte `XYCardTerminal`, die Subklassen der abstrakten Basisklasse `CardTerminal` für alle Kartenschreiblesegeräte sind.

Kartendienste

Um auf Dienste der Chipkarte zugreifen zu können, werden Diensttreiber durch den Einsatz des objektorientierten Entwurfsmusters abstrakte Fabrik erzeugt. Eine konkrete Fabrik wird verwendet, um Diensttreiber für alle Chipkarten eines Herstellers zu erzeugen. Diese konkrete Fabrik erkennt den Typ der Karte anhand ihrer Antwort auf einen Resetpuls und instantiiert daraufhin Treiber für genau diesen Kartentyp.
Um für eine beliebige Karte die entsprechenden Treiber automatisch instantiieren zu können,

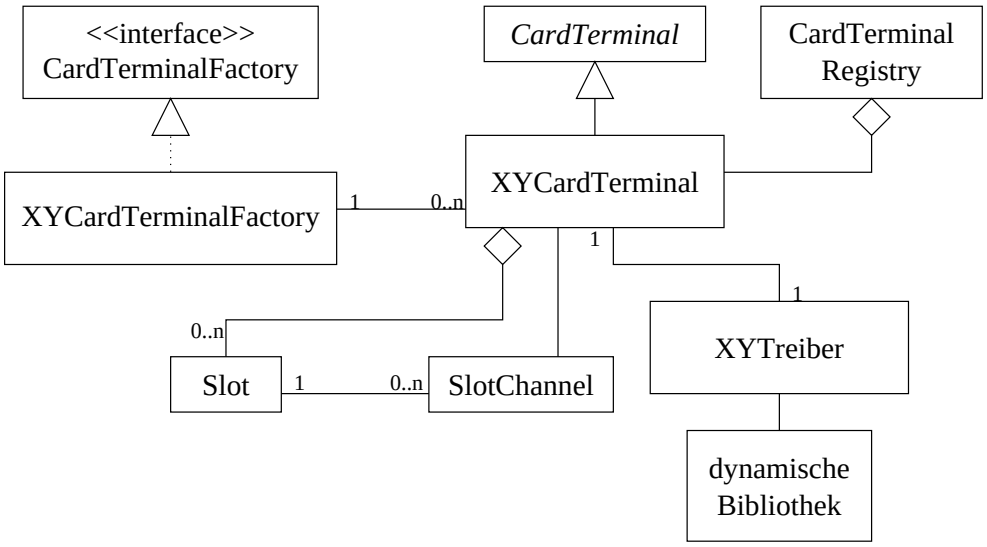

Abbildung 2.18: Klassendiagramm für das Kartenschreiblesegerät

wird jede konkrete Kartendienstfabrik bei einer globalen Registrierung (`CardServiceRegistry`) angemeldet. Werden Treiber für eine bestimmte Karte benötigt, überprüft die Registrierung anhand der Antwort auf einen Resetpuls, ob eine der konkreten Fabriken für diese Karte Treiberklassen erzeugen kann. Es wird die konkrete Fabrik zur Treibererzeugung verwendet, die diese Karte unterstützt. Ob ein bestimmter Diensttreiber für eine eingelegte Karte existiert, können Objekte der Klasse `SmartCard` überprüfen.

Ausnahmebehandlung

Falls die Ausführung einer Methode fehlschlägt, z.B. weil während eines Kartenzugriffs die Chipkarte aus dem Lesegerät entfernt wird, wird eine Ausnahme ausgelöst. Im OpenCard-System existiert eine Hierarchie von Ausnahmeklassen, die für alle Fehler, die auftreten können, vordefinierte Ausnahmeklassen enthält.

Konfiguration des Systems

Da JAVA auf verschiedensten Plattformen lauffähig ist und Code über die Netzwerkschnittstelle von JAVA nachgeladen werden kann, ist im System selbst kein Konfigurationsmechanismus festgeschrieben: Es existiert eine Schnittstelle (`OpenCardConfigurationProvider`), die eine Klasse zum Laden von Konfigurationsdaten implementieren muß. Mit Hilfe einer solchen Klasse kann OPENCARD die Konfigurationsdaten laden.
Wird die Konfiguration, wie auf den meisten Rechnern üblich, aus einer Datei geladen, können Konfigurationsinformationen durch eine mitgelieferte Klasse geladen werden. Dieser Lader implementiert die Schnittstelle `OpenCardConfigurationProvider`. Für Netzwerkcomputer ohne eigene Festplatte ist es erforderlich, eine Klasse zu implementieren, die die Konfigurationsdaten über das Netzwerk lädt.
Soll die Konfiguration des Systems aus einer Datei geladen werden, wird der Konfigurationsdateilader (`OpenCardPropertyFileLoader`) verwendet. Das Format für die Konfigurations-

datei entspricht dem Format anderer Konfigurationsdateien (vergleiche [Eckel 98]) für die JAVA Umgebung.
Die Konfigurationsdatei wird dann beim Start des OPENCARD-Systems (`SmartCard.start()`) geladen und ausgewertet.
Konfigurationen werden für konkrete Dienstfabriken und Fabriken für Schreiblesegerättreiber durchgeführt (Beispiel in Anhang A.2 auf Seite 73).

Kapitel 3

Objektorientierte Modellierung von Chipkartenfunktionen

Der Entwurf des objektorientierten Modells für Chipkartenfunktionen wird mit einem Standardverfahren der objektorientierten Entwicklung durchgeführt [Oesterreich 97]. Zunächst wird als Vorstudie eine Einordnung der Chipkarte nach softwaretechnischen Gesichtpunkten vorgenommen (siehe Abschnitt 3.1), um typische Merkmale von Chipkarten herauszuarbeiten.
Da ein Modell für Chipkarten entwickelt werden soll, wird in der Anforderungsanalyse (vergleiche Abschnitt 3.2) definiert, wie das Modell entworfen wird.
In der Anwendungsfallanalyse (siehe 3.3) werden zwei verschiedene Szenarien für die Benutzung von Chipkarten auf ihre Anwendungsfälle hin untersucht.
Aus den so gefundenen Anwendungsfällen wird ein Modell für die Funktionen von Chipkarten entwickelt, das Modell der Dienste (siehe Abschnitt 3.4). Das objektorientierte Modell zur Benutzung von Chipkartenfunktionen setzt auf das Modell der Dienste auf (siehe Abschnitt 3.5)

3.1 Einordnung von Chipkarten unter softwaretechnischen Gesichtspunkten

Um eine Einordnung der Chipkarte aus Sicht eines Softwareentwicklers vornehmen zu können, wird ein Vergleich der Chipkarte und des integrierten *card operating systems* mit heute üblichen Datenträgern (Diskette und Festplatte) und einem einfachen Netzwerkcomputersystem [Sun 98c] vorgenommen (siehe Tabelle 3.1 auf der nächsten Seite[1]).
Vergleichskriterien, nach denen die einzelnen Medien verglichen werden, sind:

- **Persistenz:** Ermöglicht das Medium langlebige Speicherung von Daten auch über ein Abschalten der Versorgungsspannung hinaus?
- **Mobilität:** Ist das Medium physikalisch mobil?

[1]Die Einordnung soll nicht darüber hinwegtäuschen, daß Chipkarten weder über soviel Speicher verfügen wie Disketten oder gar Festplatten, noch daß Rechenleistungen von Computersystemen vorhanden wären. Da Chipkartentechnologie noch sehr jung ist, sind große Leistungssteigerungen bei Speicherkapazität und Ausführungsgeschwindigkeit zu erwarten.

	Chipkarte	Diskette	Festplatte	Netzwerk-computersystem
Persistenz	Ja	Ja	Ja	Nein
Mobilität	Ja	Ja	Nein	Nein
Sicherheitssystem	aktiv/integriert	passiv	passiv	aktiv/aufgesetzt
Funktionen/RPC	Ja	Nein	Nein	Ja
Funktionen/REV	Teilweise (JAVACARD)	Nein	Nein	Teilweise (TYCOON, JAVA...)

Tabelle 3.1: Eigenschaften von Chipkarten und anderen Medien

- **Sicherheitssystem:** Ist es möglich, den Zugriff auf Daten zu schützen?
- **Funktionsausführung:** Erlaubt das Medium das Ausführen von Funktionen?

Ziel dieser Aufstellung ist es, einen Vergleich der Funktionalitäten zu ermöglichen. Das mobile Medium Chipkarte ermöglicht langlebige und geschützte Speicherung von Daten wie Disketten. Auf der anderen Seite führen Chipkarten aktiv Funktionen aus (z.B. einen Verschlüsselungsalgorithmus), eine Eigenschaft über die Rechnersysteme verfügen, die entfernte Funktionsaufrufe (*remote procedure calls*, RPCs) ausführen können.
Chipkarten, die über Funktionsausführungsmöglichkeiten verfügen, z.B. JAVA-Karten, können zusätzlich übermittelten Code ausführen (*remote evaluation*, REV).
Chipkarten werden in persistenten und verteilten Systemen eingesetzt. Sie werden sowohl zur Datenspeicherung als auch zur Authentifizierung und Autorisierung von Benutzern gegenüber einem Hintergrundsystem eingesetzt. Die kombinierten Anforderungen an Persistenz und Sicherheit entsprechen den Anforderungen an das Hintergrundsystem. Die Sicherheit der Benutzerdaten kann auf diese Weise weder im Hintergrundsystem noch auf der Chipkarte des Benutzers kompromittiert werden.
Die Hintergrundsysteme sind persistente Systeme die mit großen Datenmengen arbeiten können. Daten im Sinne einer persistenten Programmierumgebung sind sowohl reine Archivdaten (passive Daten), als auch Funktionen zum Umgang mit Daten (aktivierbare Objekte). Um solche Systeme miteinander interagieren zu lassen, existieren Lösungen [Mathiske 96; Johannisson 97], die es ermöglichen Prozesse (aktive Objekte), auf andere Systeme migrieren zu lassen. Chipkarten mit Programmausführungsmöglichkeit stellen diese drei Arten von Daten zur Verfügung. Die Mobilität von Prozessen wird bei Chipkarten durch die physikalische Mobilität der Karte erreicht. Persistente Systeme sind nicht dazu in der Lage, physikalisch zu migrieren, hier wird Mobilität von Prozessen durch Verschicken von „schlafenden“ Prozeßobjekten im Netzwerk erreicht.
Unterschiedliche Typen von Chipkarten stellen Persistenz und Sicherheit auf unterschiedlichen Ebenen zur Verfügung (siehe Abbildung 3.1 auf der nächsten Seite):

- Speicherkarten haben kein Sicherheitssystem und speichern Daten binär in einem einzigen Speicherraum.
- Multifunktionskarten mit integrierten Sicherheitssystem verfügen über Möglichkeiten, Daten als Dateien abzulegen. Dateien können unterschiedliche Arten von Daten aufnehmen (Binärdaten, Datensätze und zyklische Strukturen, vergleiche Abschnitt 2.1.1). Die Daten können durch Sicherheitscodes und Verschlüsselung geschützt werden.

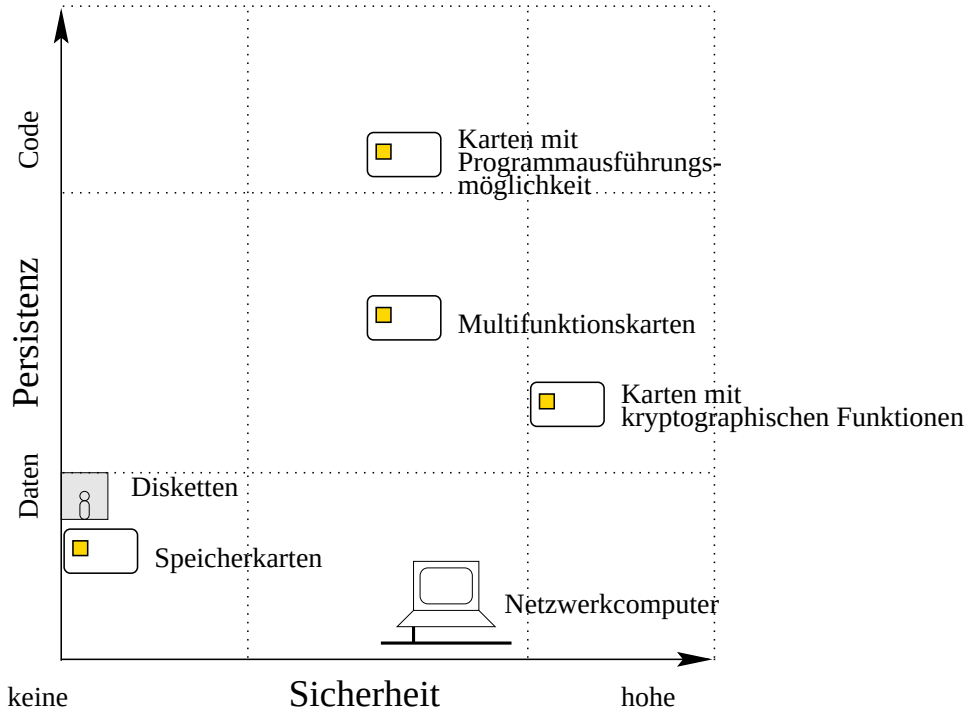

Abbildung 3.1: Orthogonalität der Chipkartenmerkmale

- Karten mit kryptographischen Funktionen erlauben ein hohes Sicherheitsniveau für die gespeicherten Daten durch die Verschlüsselung von Daten mit langen Schlüsseln. Die auf der Karte erzeugbaren Dateien sind jedoch binär strukturiert.
- Karten mit Programmausführungsmöglichkeit sind dazu in der Lage, in Kartendateien gespeicherten Code auszuführen; Der interne Status eines Kartenprogrammes wird von der Karte im nichtflüchtigen Speicher gespeichert, ist aber von außen nicht zugänglich. Das Sicherheitssystem dieser Karten entspricht dem von Multifunktionskarten.
- Disketten speichern Daten persistent in Dateien. Die Struktur der gespeicherten Daten kann nicht von der Diskette kontrolliert werden.
- Netzwerkcomputersysteme verfügen über integrierte Sicherheitsmechanismen, erlauben aber keine persistente Datenspeicherung. Im Vergleich zu Chipkarten und Disketten fehlt der Aspekt der physikalischen Mobilität.

Die Merkmale Persistenz, Ausführung von Funktionen und Mobilität sind orthogonal im Chipkartenbetriebssystem integriert, das heißt, das Sicherheitssystem ist vollständig in das Persistenzsystem (vergleiche Abschnitt 2.1.3 auf Seite 18) und die Mobilitätsunterstützung eingebunden.

3.1.1 Persistenz

Chipkarten verfügen über Datenspeicherungsmöglichkeiten. Die Langlebigkeit von Daten ist durch Verwendung von nichtflüchtigen Speichern (EEPROMs) auch bei fehlender Versor-

gungsspannung gewährleistet. Fehler in der Datenspeicherung werden im Betrieb erkannt und wenn möglich korrigiert (siehe [DeL 98b]).
Neben der Datenspeicherung ist durch die Speicherung des Kartenzustandes ein weiterer Persistenzaspekt vorhanden: Nach Abschalten der Versorgungsspannung geht der Zustand der Zugriffscodes und Sperrungen für Dateien nicht verloren.

3.1.2 Mobilität

Eine Chipkarte kann aus dem Schreiblesegerät eines Computers entfernt und an ein anderes System angekoppelt werden, ist also ein mobiles System. Die Karte kann als zwischen Systemen migrierender Datenträger verwendet werden und zusätzlich Dienste, wie die Verschlüsselung von Daten, für jedes dieser Systeme erbringen.

3.1.3 Ausführen von Funktionen

Im Gegensatz zu Disketten können Chipkarten Funktionen ausführen. Die Berechnung einer kryptographischen Funktion kann als ein *remote-procedure-call* angesehen werden. Das Schreiblesegerät veranlaßt die Karte durch einen Befehl dazu, eine komplexe Funktion auszuführen.
Sieht man die verwendeten Bytecodes als Dienstnummern, so stellt die ISO-Norm [ISO 94] standardisierte Dienstnummern und Parameter zur Verfügung. Die verwendbaren 2 Bytes langen Codes für Fehlermeldungen sind ebenfalls standardisiert.
Einige Chipkarten können an sie übermittelten Code ausführen (*remote evaluation*). Dies wird beispielsweise über einen karteninternen JAVA-Bytecodeinterpreter realisiert [Sun 98b]. Dieser Interpreter kann ein JAVA-Programm, das in einer Chipkartendatei abgelegt ist, ausführen.

3.1.4 Sicherheit

Es wird unterschieden zwischen:

- Dem geschütztem Produktionsprozeß, der unerlaubte Manipulation der Karte während der Herstellung verhindert und
- Sicherung von Kartendaten während der Benutzung der Karte.

Durch beide Verfahren wird ein hohes Sicherheitsniveau erlangt.

Sicherheit durch Schlüssel

Chipkarten verfügen je nach Kartentyp über unterschiedlich stark ausgeprägte, aktive Sicherheitssysteme, die unerwünschten Zugriff auf Daten verhindern. Alle Kartenfunktionen, die auf Daten zugreifen, sind durch dieses System geschützt. Schutz von Daten ist auf zwei verschiedene Arten möglich:

- Schutz durch einen Zugriffscode
- Schutz durch Verschlüsselung der Daten

Beide Arten des Schutzes stellen sicher, daß nur derjenige, der über einen entsprechenden Code verfügt, auf die Daten zugreifen kann. Beim Schutz durch einen Zugriffscode kommt als weiterer Aspekt hinzu, daß diese nach falscher Eingabe gesperrt werden kann und ein weiterer Zugriff auf so geschützte Kartendaten nicht mehr möglich ist (siehe Abschnitt 2.1.1 auf Seite 15).
Verschlüsselung von Daten durch die Karte bietet die Möglichkeit, Daten auf der Karte verschlüsselt abzulegen, da nur derjenige, der über den korrekten Schlüssel verfügt, die Daten entschlüsseln kann. Durch die Verwendung asymmetrischer kryptographischer Verfahren sind erweiterte Möglichkeiten vorhanden: Es können digitale Signaturen generiert werden. Durch die Verteilung von Chipkarten mit öffentlichen Schlüsseln ist es möglich, geheime Daten mit einer Karte zu verschlüsseln, ohne daß der Schlüssel preisgegeben wird. Es kann nur derjenige, der über den richtigen geheimen Schlüssel verfügt, so verschlüsselte Daten entschlüsseln.

Zugriffsschutz im Produktionsprozeß

Der Mikrocontroller, der die Kartenfunktionen ausführt ist ein möglicher Angriffspunkt um die Sicherheit von Kartensystemen in Frage zu stellen. Im Produktionsprozeß eingeschleuster Programmcode kann die gesamte Sicherheit des Systems zerstören.
Die Hersteller von Chipkartencontrollern verhindern daher, daß Daten in den Controller geschrieben werden können, ohne daß vorher ein entsprechender Schlüssel übermittelt wird. Durch Wechsel der Schlüssel in jeder Produktionsphase ist der Controller (vergleiche Abschnitt 2.1.5 auf Seite 20) sicher vor entsprechenden Angriffen.
Durch dieses Verfahren kann für eine Karte, die den Endverbraucher erreicht, garantiert werden, daß auf den Transportwegen keine Manipulation stattgefunden hat. Über gleichwertige Schutzmechanismen während des gesamten Produktionsprozesses verfügt kein anderes Computersystem.

3.2 Anforderungen an das Modell

Geeignete Kriterien für den Entwurf eines objektorientierten Modells ergeben sich aus den Forderungen eines Programmierers, der Chipkartenanwendungen entwickelt:

- **Einheitliche Programmierschnittstellen** für Karten mit ähnlichen Merkmalen:

  Durch identische Schnittstellen zu verschiedenen Chipkarten können für eine Karte entwickelte Anwendungen leicht für andere Karten portiert werden. Die Entwicklungszeit für neue Anwendungen verkürzt sich, da eine andere Karte über die gleiche Programmierschnittstelle angesprochen wird. Der Einsatz in verteilten Systemen wird hierdurch vereinfacht.

- **Austauschbarkeit** der Treiber für verschiedene Karten, um neue Karten mit wenig Aufwand integrieren zu können:

  Beim Entwurf des Modells wird auf Plattformunabhängigkeit geachtet. Eine plattformunabhängige Programmiersprache (JAVA) wird verwendet. Treiber für Hardware werden so modelliert, daß eine einfache Austauschbarkeit gewährleistet ist. Der hardwareabhängige Teil wird stets so gekapselt, daß nur ein minimaler Teil des Modells und des daraus resultierenden Codes hiervon abhängig ist.

- **Verwendung** von existierenden Standards, um eine große Zahl von Karten zu unterstützen:

  Es werden bestehende Standards verwendet, um Chipkarten anzusprechen. Das entworfene Modell kann auf diese Weise für die verschiedenen Einsatzbereiche von Chipkarten verwendet werden. Unterschiedliche Karten sollen mit wenig Aufwand zu integrieren sein. Schnittstellen werden so entworfen, daß Karten mit ungebräuchlichen Funktionsmerkmalen einfach zu integrieren sind. Wo immer es möglich ist, werden Funktionen betrachtet, die in der ISO Norm festgelegt sind. Die Verwendung von nicht standardisierten Kartenfunktionen bildet stets die Funktionalität standardisierter Funktionen nach.

Die Schnittstellen werden analog zu Schnittstellen auf anderen Computersystemen konzipiert: Der Treiber für das Chipkartendateisystem erlaubt Zugriff auf Chipkartendateien über eine Schnittstelle, die der einer „normalen“ Datei ähnlich ist. Hieraus resultiert eine kurze Einarbeitungszeit für neue Benutzer.
Die Nutzung von Chipkarten in verteilten Systemen wird im Rahmen dieser Arbeit durch Verwendung der Programmiersprache JAVA gelöst, die Verteilung von Aktivitäten in Computernetzwerken unterstützt. JAVA DATABASE CONNECTIVITY [Sun 98d] ermöglicht den Zugriff auf persistente Systeme (Datenbanken), die über SQL (*structured query language*) angesprochen werden.

3.3 Anwendungsfallanalyse

In dieser Arbeit wird ein Softwaremodell für die Benutzung von Chipkarten entworfen. Zwei Anwendungsszenarien für Chipkarten für zwei Phasen des Chipkartenlebens (vergleiche Abschnitt 2.1.5 auf Seite 20) werden betrachtet:

- Die **Personalisierungsphase**, in der die Karte für den Benutzer verfügbar gemacht wird.
- Die **Benutzungsphase**, in der sich die Chipkarte den größten Teil der Lebensdauer befindet.

Bevor die Chipkarte personalisiert werden kann, durchläuft sie einen komplexen Herstellungsprozeß, dessen Anwendungsfälle hier nicht betrachtet werden. Nur die Benutzung von Chipkarten wird untersucht. Die Lebensende-Phase der Chipkarte, die durch mechanische Abnutzung des Kontaktfeldes der Karte oder durch Sperrung der Karte vom COS erreicht wird, braucht nicht separat modelliert zu werden. Bei mechanischer Abnutzung ist keine Benutzung der Karte durch Zugriff auf Daten in der Karte mehr möglich. Die Sperrung der Karte durch das COS wird von den für die Benutzungsphase entworfenen Routinen detektiert.
Die im Rahmen dieser Arbeit modellierten Phasen sind die, in denen der Datenaustausch mit Chipkarten über an einen Rechner angeschlossene Schreiblesegeräte stattfindet.
Zwei verschiedene Anwendungsszenarien, die der Personalisierungsphase und der Benutzungsphase entsprechen, werden betrachet. Die Verwendung der Chipkarte erfolgt durch einen Benutzer und die Personalisierung der Chipkarte durch den Herausgeber. Die Anwendungsfälle betrachten die Funktionalität der Chipkarte, die eine Applikation benötigt. Sie befinden sich von der Granularität her zwischen Anwendungsfällen für eine Applikation und der direkten Beschreibung der Hardware.

Anwendungsfall Personalisierung

In der Personalisierungsphase schreibt derjenige, der die Karte an den Kunden ausgibt, Daten in den Speicher der Karte. Es werden Daten geschrieben, die während der Benutzung der Karte nur noch gelesen werden dürfen:

- Sicherheitscodes der Karte werden eingerichtet,
- Unveränderbare Daten des Herausgebers und des Kartennutzers werden geschrieben,
- Geheime Schlüssel für kryptographische Algorithmen werden geschrieben.

Um in die Herstellungsphase abzuschließen, wird eine irreversible Sperre geschrieben. Es werden drei Anwendungsfälle betrachtet (siehe Abbildung 3.2).

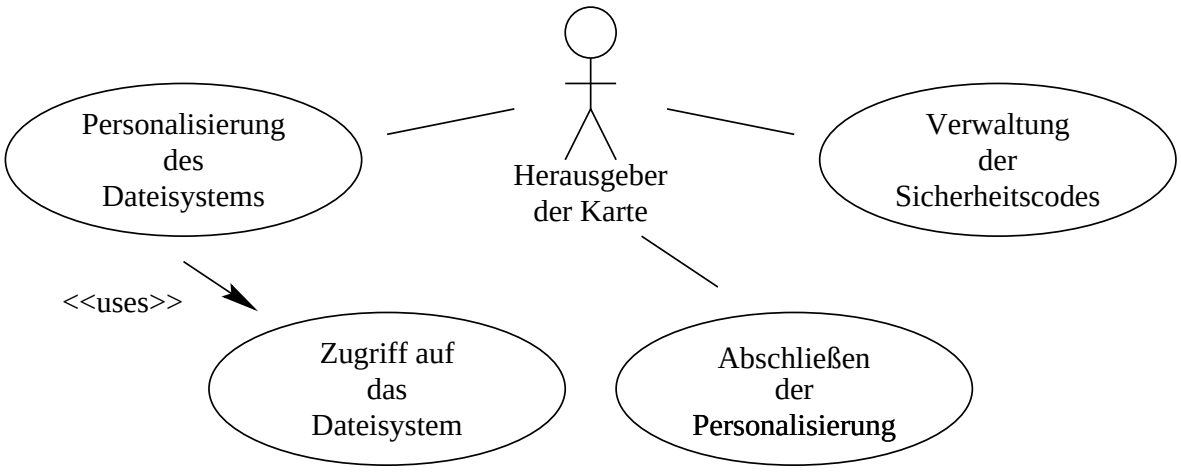

Abbildung 3.2: Anwendungsfälle für den Herausgeber

- **Personalisierung des Dateisystems:** Die zur Benutzung der Karte erforderlichen Verzeichnisse und Dateien werden erzeugt und die erforderliche Zugriffsrechte vergeben.
- **Zugriff auf das Dateisystem:** Zugriff auf Chipkartendateien ist erforderlich, um Daten in den Speicher der Chipkarte zu schreiben.
- **Verwaltung der Sicherheitscodes:** Sicherheitscodes werden in den entsprechenden Speicherbereichen eingetragen. Die Informationen, mit welchen Sicherheitscodes welche Funktionen ermöglicht werden, werden geschrieben.
- **Abschließen der Personalisierung:** Die Personalisierungsphase wird durch Schreiben einer Sperre irreversibel abgeschlossen. Die Karte geht damit in die Benutzungsphase über und die Sicherheitscodes werden aktiv.

Anwendungsfall Benutzung

Der Benutzer verwendet die Karte und benötigt Zugriff auf Daten und Funktionen. Normalerweise sind für den Benutzer diese Zugriffe nicht sichtbar, da sie von der jeweiligen Anwendung ausgeführt werden. Kartenfunktionen und -daten werden nur indirekt von menschlichen Benutzern verwendet. Der Zugriff wird durch ein Anwendungsprogramm vorgenommen. Abbildung 3.3 zeigt die Anwendungsfälle für die Benutzungsphase der Chipkarte. Die auftretenden Anwendungsfälle sind:

- **Zugriff auf Daten der Karte:** Die Anwendung benötigt auf der Karte gespeicherte Daten.
- **Zugriff auf Funktionen der Karte:** Funktionen der Karte können von der Anwendung aufgerufen werden. Es sind verschiedene Gruppen von Funktionen auf Chipkarten vorhanden, die jeweils einzelnen Anwendungsfällen entsprechen.
  - **Zugriff auf kryptographische Funktionen:** Kryptographische Funktionen werden verwendet, um Chipkarten zu authentifizieren, Daten zu verschlüsseln und digitale Signaturen zu berechnen.
  - **Zugriff auf weitere Funktionen:** Einige Karten verfügen über vielfältige Funktionen: Zähler und Zufallszahlengeneratoren sind einfache Funktionen, Karten mit JAVA-Bytecodeinterpreter können geladene Programme ausführen. Die Verwendung solcher Funktionen entspricht jeweils einem Anwendungsfall.
- **Zugriff auf Sicherheitscodes:** Der Benutzer muß Sicherheitscodes eingeben, um geschützte Funktionen der Karte verwenden zu können. Funktionen zum Verändern von Zugriffscodes gehören ebenfalls zu diesem Anwendungsfall.

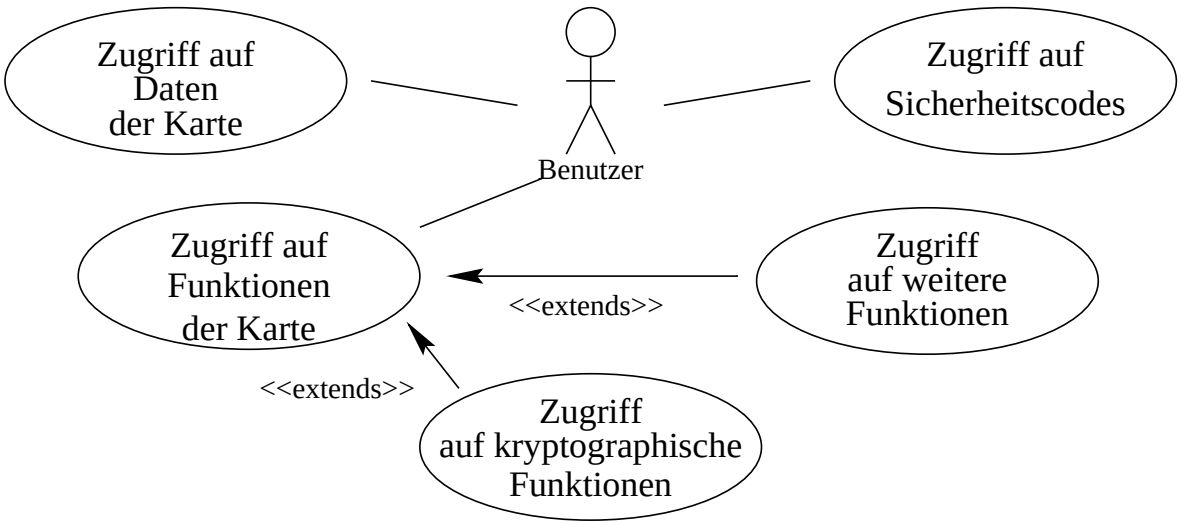

Abbildung 3.3: Anwendungsfälle für den Applikationsentwickler

Der Benutzer bekommt Zugriff auf Daten und Funktionen der Chipkarte durch ein Kartenschreiblesegerät. Dieses wird in der Anwendungsfallanalyse nicht betrachtet. Ein objekt-

orientiertes Modell der Treiber für Kartenschreiblesegeräte wird in Abschnitt 3.6 auf Seite 49 vorgestellt.

3.4 Dienstleistungen als Modell für Chipkartenfunktionen

Die Chipkarte ist ein aktives Medium, das komplexe Funktionen wie z.B. einen kryptographischen Algorithmus ausführen kann. Ein Modell für diese von der Chipkarte angebotenen Funktionen ist die **Dienstleistung**. Die Karte erbringt einen Dienst für das Computersystem, an das sie gerade angebunden ist.
Der Zugriff auf einen Dienst findet mit Hilfe eines Diensttreibers statt. Für den Diensttreiber ist die Chipkarte der Dienstleister. Aus Sicht des Anwendungsprogrammes, das ausschließlich über den Diensttreiber mit der Karte kommuniziert, erscheint der Diensttreiber als Dienstleister.

Klassifikation von Diensten Die Anwendungsfallanalyse zeigt, daß es verschiedene Anwendungsfälle gibt, die auf unterschiedliche Funktionen der Chipkarte zugreifen. Einzelne Funktionen der Karte sind verschiedenen Anwendungsfällen zugeordnet.
Um aus den allgemein gehaltenen Anwendungsfällen verwendbare Dienste zu gewinnen, werden die einem Anwendungsfall zugehörigen Funktionen der Chipkarte einzelnen Diensten zugeordnet. Ein Anwendungsfall kann über eine einzige Funktion verfügen oder mehrere Funktionen zur Diensterbringung verwenden. Alle Funktionen eines Anwendungsfalles gehören zum gleichen Dienst. In Abbildung 3.4 sind beispielhaft Dienste aufgeführt, die den im vorigen Abschnitt betrachteten Anwendungfällen entsprechen.

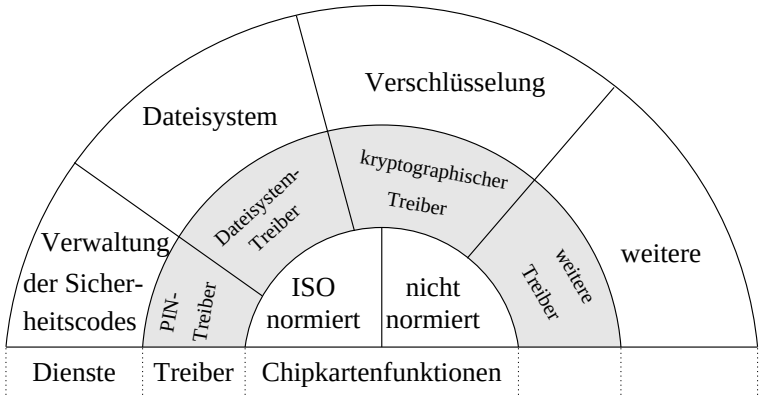

Abbildung 3.4: Kapselung der Chipkartenfunktionen durch Diensttreiber

Ein Dienst wird dem Anwendungsprogramm durch einen Diensttreiber angeboten:

- Zugriff auf Daten einer Chipkarte wird durch den Dateisystemdienst der Chipkarte zur Verfügung gestellt.
- Zugriff auf kryptographische Funktionen wird durch den kryptographischen Dienst bereitgestellt.
- Zugriff auf Sicherheitscodes wird vom Zugriffscode-Verwaltungsdienst durchgeführt.

- Für den Zugriff auf weitere Funktionen der jeweils betrachteten Karte werden weitere Dienste und zugehörige Diensttreiber benötigt.

Einige Dienste sind aus standardisierten Chipkartenfunktionen zusammengesetzt und damit auf allen Chipkarten ähnlich, andere Dienste sind nicht oder nur teilweise aus standardisierten Funktionen aufgebaut. Zu den standardisierten Diensten zählt der Dateisystemdienst. Ein nicht standardisierter Dienst ist der Verschlüsselungsdienst, der die kryptographischen Funktionen der Karte verfügbar macht.
Die Aspekte der Persistenz, Sicherheit und Mobilität werden ins Konzept der Dienstleistungen integriert:

- Die **Persistenz** der Kartenzustände wird im Diensttreiber nachgebildet. Der Status der Zugriffscodes wird vom Treiber gespeichert, bei gesperrtem Zugriffscode wird kein weiterer Zugriff auf geschützte Dateien durchgeführt.
- Das integrierte **Sicherheitskonzept** der Karte wird in die Diensttreiber aufgenommen. Zugriffscodes werden nicht von den Treibern gespeichert.
- Die **Mobilität** der Karte macht sich für einen Dienst durch Ausbleiben einer Kartenantwort bemerkbar. Dies wird durch eine Ausnahme signalisiert. Eine Ereignisverwaltung benachrichtigt Programme, wenn eine Chipkarte in ein Schreiblesegerät eingelegt wird.

Für jeden der Anwendungsfälle wird ein Diensttreiber entwickelt. In den Diensttreibern werden keine Anwendungsszenarien mehr unterschieden; daher finden sich teilweise zwei Anwendungsfälle bei einem Dienst.

- **Personalisierung des Dateisystems** und **Abschließen der Personalisierung** werden durch einen Personalisierungsdienst gelöst.
- **Zugriff auf das Dateisystem** und **Zugriff auf Daten der Karte** werden vom Dateisystemdienst durchgeführt.
- **Verwaltung der Sicherheitscodes** und **Zugriff auf Sicherheitscodes** werden vom der Zugriffscodeverwaltung übernommen.
- **Zugriff auf Funktionen der Karte:**
  - **Zugriff auf kryptographische Funktionen** werden durch den kryptographischen Dienst verfügbar gemacht.
  - **Zugriff auf weitere Funktionen** muß durch Diensttreiber, die den Funktionen entsprechen, durchgeführt werden.

3.5 Ein objektorientiertes Modell für Diensttreiber

Übermittelte Befehle sind kartenspezifisch für die beschränkten Betriebsmittel der Karte entworfen. Sie sind nicht dazu geeignet, direkt die Programmierschnittstelle für eine Hochsprache darzustellen.
Wie im vorigen Abschnitt gezeigt, können die elementaren Funktionen der Chipkarte zu verschiedenen Diensten gruppiert werden. Zusammengehörige elementare Funktionen werden im entworfenen Modell durch einen Diensttreiber gekapselt. Leitlinien bei der Entwicklung eines Diensttreibers sind:

- Die **Funktionalität** des Dienstes wird durch den Diensttreiber gegenüber den Protokolldatenblöcken (Bytecodes) zum Aufruf der Kartenfunktionen hervorgehoben:

  Eine Chipkartenfunktion wird nicht über ihren Bytecode aufgerufen, sondern über eine Methode, deren Name der Aufgabe der Funktion entspricht. Anstelle einen Bytecode zur Karte zu übertragen wird eine Methode des Treibers aufgerufen. Aus dem Bytecode

  `00A40000024B5900F60000083132333435363738`$_{hex}$

  wird z.B. der Aufruf von

  ```
  cryptoService.generateCipher(new CardFilePath("KY"), 0, "12345678");.
  ```

- Die **Schnittstellen** für die Dienste werden so allgemein gehalten, daß Diensttreiber für verschiedene Karten die gleiche Schnittstelle implementieren können. Für den Programmierer werden auf diese Art Karten mit ähnlichen Funktionen identisch verwendet.

- Die nicht objektorientierten Funktionen der Karte werden in **Methoden** der Treiberklasse eingehüllt, die objektorientierte Benutzung von Chipkartenfunktionen ermöglichen. Objektorientierte Diensttreiber sind einfacher verwendbar als nicht objektorientierte Treiber, da der interne Zustand der Chipkarte gekapselt wird. Intern kann der Treiber geschwindigkeitsoptimiert auf die Chipkarte zugreifen. Die Integration neuer Treiber ist einfacher, da klar definierte Schnittstellen verwendet werden.

Die Kapselung der Dienste im Diensttreiber wird im objektorientierten Modell (siehe Abbildung 3.5) verdeutlicht.
Alle Diensttreiber haben Gemeinsamkeiten, da sie alle über ein Schreiblesegerät mit einer Chipkarte kommunizieren müssen, um Dienstanfragen durchzuführen. Diese Funktionen finden sich in der abstrakten Basisklasse für alle Diensttreiber (`CardService`). Subklassen dieser Klasse stellen die speziellen Diensttreiber für jede Chipkarte zur Verfügung. Um einheitliche

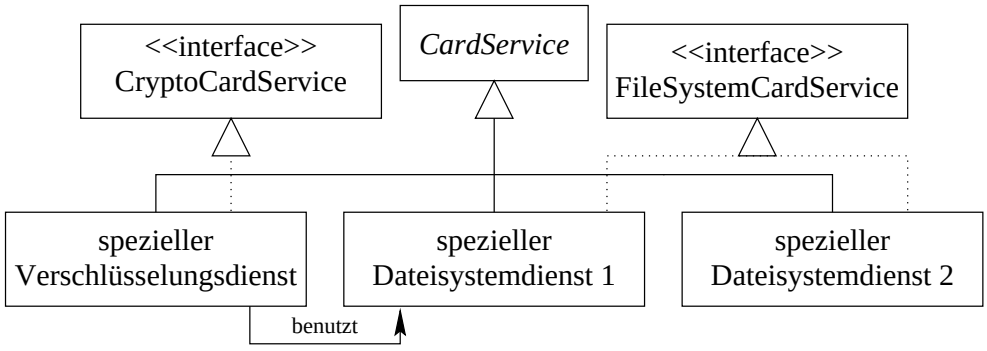

Abbildung 3.5: Treiber für die Kartendienste

Schnittstellen für diese Diensttreiber zu gewährleisten, werden den Diensten entsprechende Schnittstellen (*Interfaces*) implementiert: Die Details der einzelnen Kartenbetriebssysteme bleiben so vor dem Anwendungsprogrammierer verborgen, es wird gegen eine Schnittstelle programmiert. Spezielle Treiber für zwei verschiedene Karten können die gleiche Schnittstelle implementieren und identisch angesprochen werden. Auf Befehlsebene der Karten werden

für die beiden Karten unterschiedliche Bytecodes verwendet, ohne daß der Anwendungsprogrammierer es bemerkt.
Verfügt eine Karte z.B. über Funktionen zur Dateibehandlung, implementiert der Treiber die Schnittstelle `FileSystemCardService`. Das Dateisystem wird mit den gleichen Funktionsaufrufen angesprochen, wie Dateisysteme anderer Karten mit Treibern, die die gleiche Schnittstelle implementieren.
Für kryptographische Funktionen wird die gleiche Vorgehensweise angewendet, indem die Schnittstelle `CryptoCardService` implementiert wird.
Ein Dienst, der Funktionen eines anderen Dienstes zur Diensterbringung benötigt, kann diesen aufrufen. Im hier betrachteten Klassendiagramm ruft der Verschlüsselungsdienst den Dateisystemdienst auf, um die Datei auszuwählen, die den Schlüssel für die auszuführende kryptographische Operation enthält. Dieser Zugriff ist von außen nicht sichtbar. Es wird auf den Diensttreiber für den anderen Dienst zugegriffen, ohne daß der Dienst explizit angesprochen werden muß.
Für jeden Chipkartentyp wird ein Satz von Diensttreibern entwickelt, der alle Funktionen, über die die Chipkarte verfügt, einbindet.

Ausnutzung von Normen

Standardisierte Dienste, wie z.B. das Dateisystem nach ISO [ISO 94], werden über eine für alle Kartentypen identische Schnittstelle verwendet. Die Norm legt fest, wie eine Kartenfunktion arbeiten muß, wenn die Karte sie unterstützt.
Der Dateisystemdienst ermöglicht die Verwendung aller in der Norm vorgesehenen Funktionen. Verfügt eine Karte nicht über eine bestimmte Standardfunktion, muß sie durch Kombination anderer Funktionen ersetzt oder eine Ausnahme (*exception*) ausgelöst werden, um eine Fehlerbehandlung zu ermöglichen.
Viele Karten erlauben z.B. nur das Erzeugen einer Verzeichnisebene, während die ISO-Norm Unterverzeichnisse vorsieht. Der Dateisystemdiensttreiber für Karten ohne Unterverzeichnisse kann auf solchen Karten keine Verzeichnisse erzeugen, obwohl die Funktion im Dateisystemdiensttreiber existiert. Karten, die alle Befehle der ISO-Norm implementieren, können z.B. wenn die maximale Zahl der Unterverzeichnisse erreicht ist, die in einem Verzeichnis erzeugt werden kann, ebenfalls keine weiteren Unterverzeichnisse erzeugen. Die Fehlermeldung der Karte ohne Unterverzeichnisse meldet, daß das Erzeugen weiterer Unterverzeichnisse nicht möglich ist, um Kompatibilität zu gewährleisten. Der Versuch eines Benutzers, eine Datei in einem Unterverzeichnis der Karte anzusprechen, endet immer mit der Fehlermeldung, daß die Datei nicht existiert, da das entsprechende Verzeichnis nicht existiert.
Karten für hohe Sicherheitsanforderungen erlauben in der Benutzungsphase nicht, daß Dateien erzeugt werden. Ein Dateisystemdiensttreiber für eine solche Karte meldet, wenn die Zugriffsrechte für die Dateierzeugung abgefragt werden, daß es nicht möglich ist, weitere Dateien zu erzeugen und gewährleistet damit Kompatibilität zu anderen Treibern.

Ein Modell für einen Diensttreiber

Die nicht objektorientiert aufgebauten Funktionen der Chipkarte werden im Diensttreiber objektorientiert in das Modell integriert. Um die Funktionen der Karte anzusprechen, wird ein objektorientierter Treiber verwendet, der die Funktionsaufrufe eines Dienstes in einer Klasse

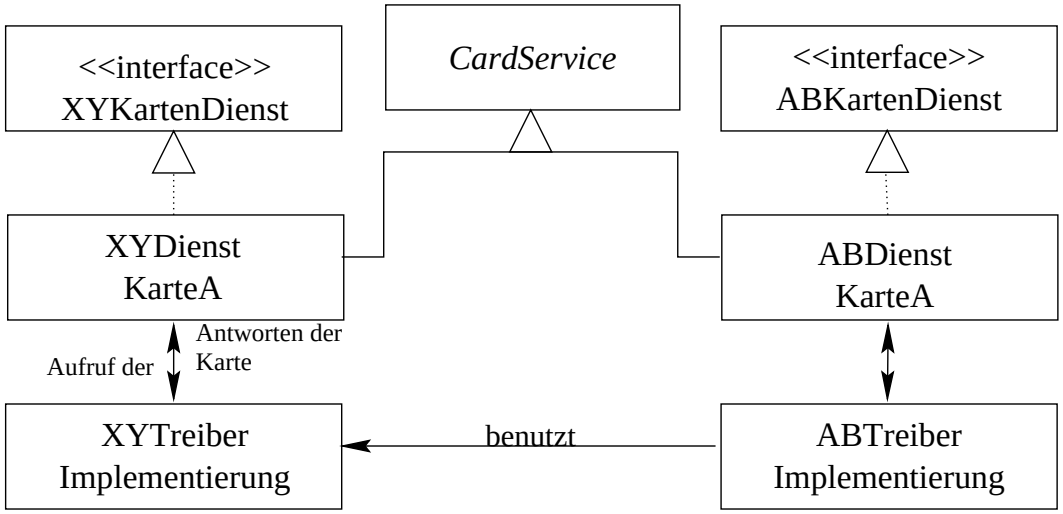

Abbildung 3.6: Zugriff auf Kartenfunktionen durch Treiber

kapselt. Die Methoden dieser Klasse werden von der generischen Treiberklasse aufgerufen, um die Schnittstelle zum Dienst zu implementieren (siehe Abbildung 3.6).
Das Klassendiagramm zeigt beispielhaft einen Treiber für den fiktiven XY-Kartendienst, Treiberklassen für eine konkrete Chipkarte werden im Abschnitt 4.4.3 auf Seite 58 erläutert. Um kompatibel mit anderen XY-Kartendiensttreibern zu sein, implementiert die Klasse `XYDienst` die Schnittstelle `XYKartenDienst`. Als Subklasse der abstrakten Basisklasse *CardService* verfügt sie über allen Diensten gemeinsame Methoden. Die Klasse `XYTreiberImplementierung` kapselt die Befehlsblöcke der Karte. In der Treiberklasse werden nur Funktionen der Klasse `XYTreiberImplementierung` zum Zugriff auf Kartenfunktionen verwendet.
Diese Struktur trennt klar zwischen den durch Bytecodes aufgerufenen Kartenfunktionen und den Diensttreibern auf höherer Abstraktionsebene. Kartendienste können auf Funktionen anderer Kartendienste zugreifen, ohne über den Umweg über Diensttreiber zu gehen. (vergleiche auch Abbildung 3.5 auf Seite 47). Die Klasse `ABTreiber`, die die Chipkartenfunktionen für die Klasse `ABDienst` kapselt, kann direkt auf die Klasse `XYTreiber` zugreifen, um Funktionen dieses Dienstes zu verwenden.

3.6 Ein objektorientiertes Modell für Chipkartenschreiblesegeräte

In der bisherigen Analyse ist das Kartenschreiblesegerät nicht betrachtet worden, da es nicht zur Modellierung der Funktionen der Chipkarte erforderlich ist. Um Daten mit einer Chipkarte auszutauschen, ist jedoch ein Kartenschreiblesegerät erforderlich. Dieser Abschnitt beschreibt das Vorgehen, Treiber für Chipkartenschreiblesegeräte auf unterschiedlichen Plattformen flexibel zu erzeugen.
Der Programmierer greift auf das Schreiblesegerät über Methoden einer abstrakten Basisklasse `CardTerminal` zu. Diese Methoden erlauben den Zugriff auf Funktionen, die alle Kartenschreiblesegeräte unterstützen, die das asynchrone ISO-Protokoll verwenden.
Allen Kartenschreiblesegeräten gemeinsame Funktionen sind in Tabelle 3.2 auf der nächsten Seite aufgeführt. Der Gerätetreiber für das Kartenschreiblesegerät ist immer hardware- und

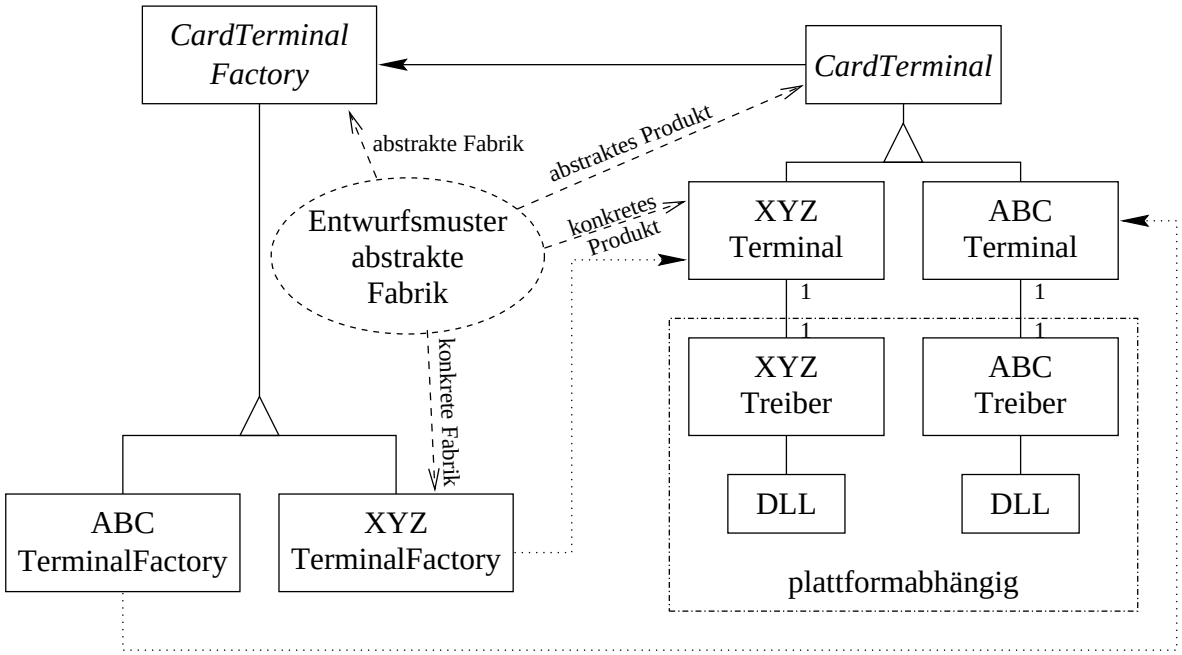

Abbildung 3.7: Treibererzeugung für Kartenschreiblesegeräte durch abstrakten Fabriken

Funktionen des Schreiblesegerätes
Spannungsversorgung für die Karte einschalten
Karte initialisieren (über einen Reset-Puls)
Befehlsblock an die Karte senden und Antwort empfangen
Spannungsversorgung für die Karte abschalten

Tabelle 3.2: Befehle, die alle Kartenschreiblesegeräte unterstützen

plattformabhängig, da ein Zugriff auf ein physikalisches Gerät stattfindet. Dieser Zugriff ist häufig über die Verwendung einer betriebssystemspezifischen dynamischen Bibliothek gelöst. Um Gerätetreiber für die einzelnen Plattformen zu instantiieren, wird das objektorientierte Entwurfsmuster der abstrakten Fabrik (*abstract factory*) [Gamma et al. 95] verwendet (siehe Klassendiagramm in Abbildung 3.7). Konkrete Subklassen einer globalen abstrakten Fabrik (hier die abstrakte Klasse `CardTerminalFactory`) instantiieren über eine Fabrik-Methode Objekte einer Subklasse der Kartenschreiblesegerät-Klasse `CardTerminal`. Jede dieser konkreten Fabriken (*concrete factories*) instantiiert Treiberklassen für Schreiblesegeräte eines anderen Herstellers (hier: `XYZTerminalFactory` instantiiert `XYZTerminal` Objekte und `ABCTerminal Factory` instantiiert `ABCTerminal` Objekte).
Um Treiber für ein anderes Kartenschreiblesegerät zu instantiieren, muß nun eine andere konkrete Schreiblesegerätetreiber-Fabrik instantiiert werden, die dann die erforderlichen Treiberobjekte erzeugt. Der Programmierer spricht alle so erzeugten Klassen über die Methoden der Basisklasse `CardTerminal` an.

Kapitel 4

Anwendungsentwicklung mit Chipkarten

Es wird beispielhaft anhand einer einfachen Anwendung das Vorgehen bei der Anwendungsentwicklung mit Chipkarten erläutert. Als Anwendung wird die automatische Anmeldung bei einem Rechnersystem mit Hilfe einer Chipkarte betrachtet.
Zunächst wird eine Anwendungsfallanalyse durchgeführt (siehe Abschnitt 4.1). Um Funktionen zu identifizieren, die die Karte beherrschen muß, wird aus den zur Chipkarte gehörenden Anwendungsfällen eine Anforderungsliste zusammengestellt. Anhand dieser Informationen wird dann eine geeignete Karte ausgewählt (siehe Abschnitt 4.2). Anhand der Informationen über die ausgewählte Chipkarte und die Anwendungsfälle (vergleiche Abschnitt 4.3) wird der Aufbau des Programmes entworfen. Die für die eingesetzte Chipkarte entwickelten Diensttreiber werden in Abschnitt 4.4 beschrieben. Auf der Chipkarte anzulegende Dateien müssen gesondert betrachtet werden, da sie zur Speicherung von geheimen Daten benutzt werden können. (siehe Abschnitt 4.5). Die Sicherheit geheimzuhaltender Daten muß durch den Einsatz von Zugriffscodes und Verschlüsselungsverfahren gewährleistet werden. Die Zugriffsrechte für die auf der Karte zu erzeugenden Dateien müssen den Sicherheitsanforderungen entsprechend gewählt werden.
Das Vorgehen ist an [Oesterreich 97] angelehnt. In einigen Punkten wird vom vorgeschlagenen Vorgehen zur Entwicklung von Software abgewichen. Chipkarten sind Hardware, die für ein Projekt angeschafft werden muß. Die Anforderungen an die Chipkarte ergeben sich aus der Anwendungsfallanalyse; nach der Anwendungsfallanalyse kann entschieden werden, welche Art von Chipkarte die für ein Projekt geeigneten Merkmale hat.

4.1 Anforderungsanalyse

Anforderungen an ein Programm zur Anmeldung bei einem Rechnersystem mit Hilfe einer Chipkarte werden zunächst spezifiziert. Im Rahmen dieser Arbeit wird der Teil der Anwendung betrachtet, der mit Chipkarten arbeitet. Für die weiteren Aufgaben wird auf frei verfügbare Programme zurückgegriffen, deren Quellen vorliegen.
Das Programm ermöglicht dem Benutzer, sich von einem MS-WINDOWS PC aus auf einer UNIX-*workstation* anzumelden; Benutzername und Paßwort werden auf einer Chipkarte gespeichert. Wenn die Chipkarte in ein an den Rechner angeschlossenes Kartenschreiblesegerät eingelegt wird, wird automatisch eine *telnet*-Verbindung zu einem vorher festgelegten Rech-

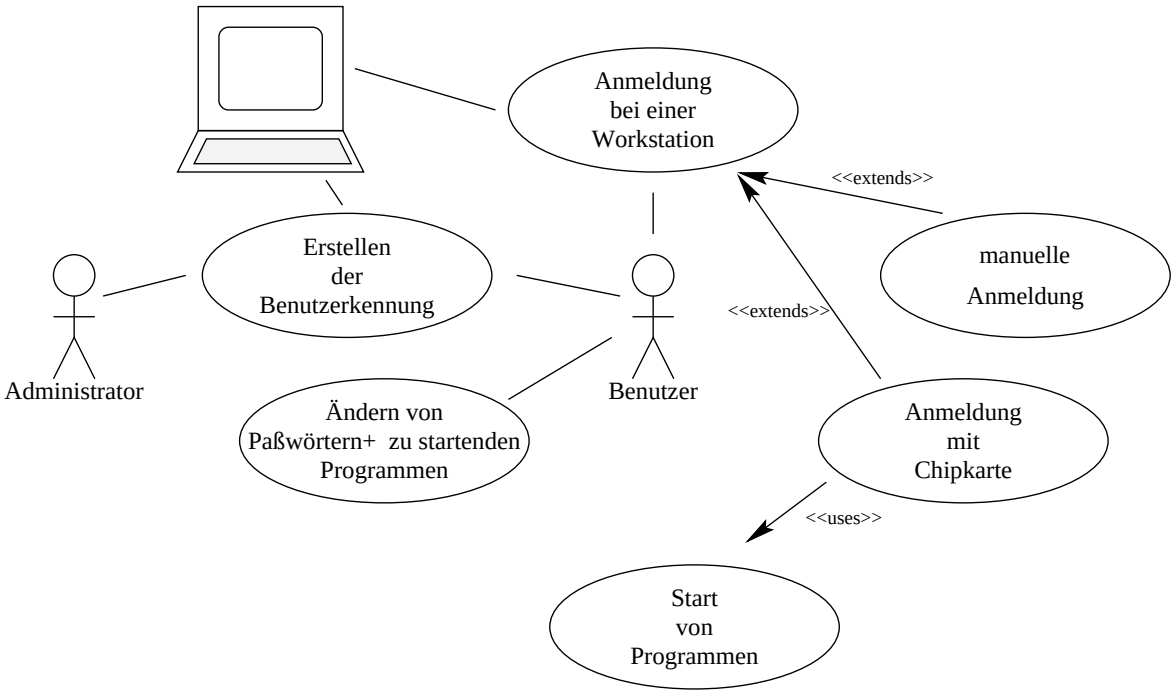

Abbildung 4.1: Rechneranmeldung mit Chipkarte

ner aufgebaut. Die Anmeldung beim anderen System wird ohne Interaktion des Benutzers vorgenommen. Benutzername und Paßwort werden an das andere System übermittelt. Um Mißbrauch zu verhindern wird das Paßwort verschlüsselt auf der Karte abgelegt.
Die Verbindung wird abgebaut, wenn die Chipkarte entfernt wird.
Als Erweiterung sollen die vom Benutzer bevorzugten Programme automatisch gestartet werden. Die benötigten Informationen werden ebenfalls auf der Chipkarte abgelegt.
Abbildung 4.1 zeigt das Anwendungsfalldiagramm für diese Anwendung. Die oben genannten Anwendungsfälle „manuelle Anmeldung bei einer UNIX-*workstation*“ oder „mit Chipkarte“, „Automatisches Starten von Programmen“ und „Auswahl der zu startenden Programme“ werden in Abschnitt 4.3 detaillert erläutert, da für eine detaillierte Analyse eine Betrachtung der verwendeten Chipkarte erforderlich ist.
„Erstellen der Benutzerkennung“ stellt einen Sonderfall dar, da nur der Administrator Benutzerkennungen erstellen und Chipkarten für Benutzer personalisieren kann. Die Karte wird in der Personalisierungsphase vom Administrator initialisiert. Eine irreversible Sperre wird geschrieben, bevor die Karte an den Benutzer ausgegeben wird.

4.2 Auswahl einer geeigneten Chipkarte

Verfügbar sind die in Abschnitt 2.1.7 auf Seite 25 genannten Kartentypen: Multifunktionskarten, Karten mit kryptographischen Funktionen und Identifikationskarten.
Die Forderung, die Anmeldung direkt nach dem Einlegen, ohne weitere Interaktion des Benutzers durchzuführen, führt dazu, daß eine Karte mit kryptographischen Funktionen verwendet werden muß, um das Paßwort des Benutzers zu verschlüsseln. Die Verwendung eines asym-

metrischen Verfahrens bietet sich an. Das Paßwort kann mit dem öffentlichen Schlüssel von der Karte verschlüsselt werden; nur die Anwendung verfügt über den geheimen Schlüssel, der das Paßwort entschlüsselt.
Auf der Karte müssen Benutzername, Paßwort und die Liste der beim Anmelden zu startenden Programme gespeichert werden. Damit das Paßwort auf der Chipkarte nicht unbeabsichtigt verändert werden kann, wird das Schreiben des Paßwortes mit einer 4-stelligen, vom Benutzer veränderbaren persönlichen Identifikationsnummer (PIN) geschützt.
Es handelt sich um eine Anwendung mit niedrigem Sicherheitsniveau für den kryptographischen Algorithmus, da ein Verlust der Karte vom Benutzer schnell bemerkt wird und das Paßwort für die Anmeldung daraufhin geändert werden kann. Eine RSA-Schlüssellänge von 512 Bits ist daher ausreichend.
Die DELARUE DXPLUS-Karte kann den erforderlichen Algorithmus ausführen und verfügt außerdem über 8 KBytes Speicher für Daten. Dieser Speicher steht für Konfigurationsdaten von Programmen zur Verfügung, die nach der Anmeldung auf der UNIX-*workstation* gestartet werden.
Die DXPLUS-Karte erfüllt die Anforderungen für die betrachtete Anwendung.

4.3 Entwurf der Anwendung

Um den Programmaufbau zu spezifizieren, werden die in Abbildung 4.1 auf der vorherigen Seite gezeigten Anwendungsfälle einzeln betrachtet, die direkt mit der Chipkartenprogrammierung verbunden sind:

Anmeldung bei einer UNIX-*workstation* mit Hilfe einer Chipkarte: Vom MS-WINDOWS-PC wird eine *telnet*-Verbindung zu einer UNIX-*workstation* aufgebaut. Benutzername und das verschlüsselte Paßwort werden aus Chipkartendateien gelesen und zum Anmelden beim anderen System verwendet. Abbildung 4.2 zeigt den Aufbau der Anwendung aus verschiedenen Paketen.

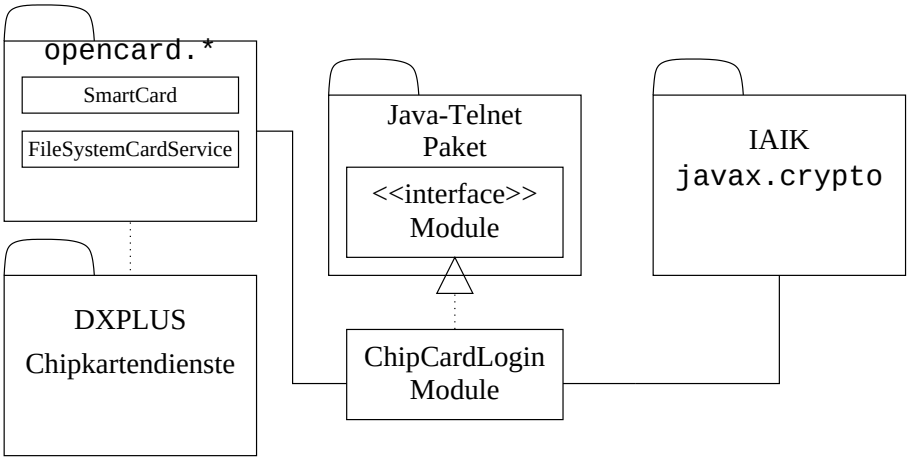

Abbildung 4.2: Aufbau der Anwendung

Um die *telnet*-Verbindung zum anderen System aufzubauen, wird das frei verfügbare JAVA-Telnet von Matthias Jugel und Marcus Meißner [Jugel, Meißner 97] verwendet
Das Entschlüsseln des verschlüsselten Paßwortes auf dem PC wird mit dem JAVA Paket `javax.crypto` der Technischen Universität Graz [Platzer 98] gelöst. Für universitären Gebrauch ist dieses Paket frei verfügbar.
Der Anwendungsfall wird als Modul für das JAVA-Telnet *applet* entwickelt und kann so einfach bestehende Anwendungen, die das *telnet-applet* verwenden, erweitern.
Mehrere Teilaufgaben werden abgearbeitet, sobald eine Chipkarte in das Schreiblesegerät eingelegt wird. Der Ablauf des Anmeldevorganges wird folgendermaßen durchgeführt:

1. Es wird überprüft, ob alle zum Anmelden erforderlichen Daten (Benutzerkennung und Paßwort) vorhanden sind.
2. Die Daten werden gelesen.
3. Die Verbindung zum entfernten System wird aufgebaut.
4. Die Benutzerkennung wird übertragen.
5. Das Paßwort wird entschlüsselt.
6. Das Paßwort wird übertragen.
7. Die vom Benutzer bevorzugten Programme werden gestartet.

Die Verbindung wird abgebaut, sobald die Chipkarte entfernt wird. Die Übertragung des Paßwortes wird unverschlüsselt durchgeführt, da das *telnet*-Protokoll es so vorsieht.

Start der vom Benutzer bevorzugten Programme: Falls das Anmelden beim Zielsystem erfolgreich war, werden die vom Benutzer bevorzugten Programme gestartet; hierzu wird der Inhalt einer Chipkartendatei ausgelesen und auf dem Zielsystem als *script* ausgeführt:

1. Der Inhalt der Präferenzen-Datei auf der Karte wird gelesen und
2. der Inhalt dieser Datei wird direkt dem im Zielsystem gestarteten Kommandozeilen-Interpreter übergeben und von diesem ausgewertet.

Für die hier beschriebene Anwendung wird die Anmeldung bei einem UNIX-System vorgenommen: Der verwendete Kommandozeilen-Interpreter ist die *c-shell* `csh`.

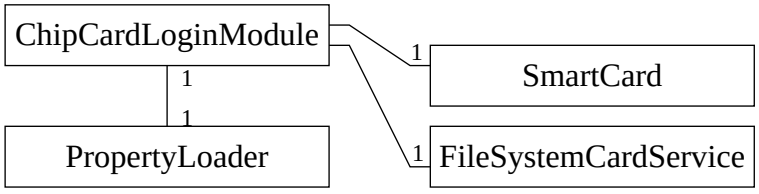

Abbildung 4.3: Klassendiagramm der Ladeklasse für Benutzerpräferenzen

Der Anwendungsfall wird in eine Klasse codiert (vergleiche Abbildung 4.3), die vom `ChipCardLoginModule` verwendet werden kann. Um Daten aus der Chipkartendatei zu lesen, wird der Dateisystemdienst der Karte verwendet.

Ändern von Paßwörtern und zu startenden Programmen: Ein JAVA-*applet* (siehe Abbildung 4.4), das den Dateisystemdienst und den kryptographischen Dienst der Chipkarte nutzt, wird verwendet, um das gespeicherte Paßwort und die Datei mit den zu startenden Programmen zu verändern. Im Prototyp der Anwendung kann ein vorher mit einem Editor erstelltes

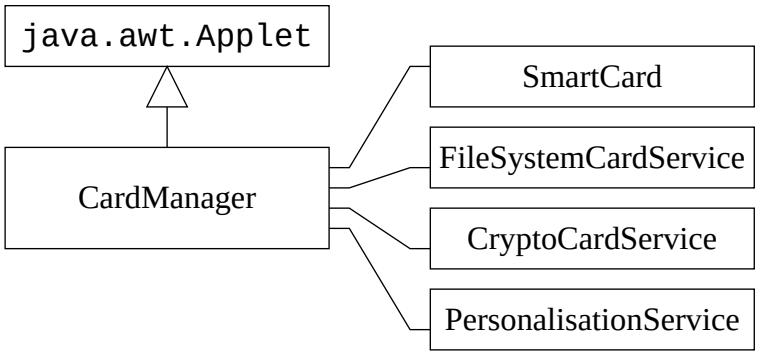

Abbildung 4.4: Klassendiagramm für die Kartenverwaltung

shell script zum Start von Programmen auf der Karte gespeichert werden.
Das Applet muß folgende Aufgaben erfüllen:

- Einlesen von Paßwort und Dateinamen der Präferenzendatei.
- Einlesen der Präferenzendatei.
- Verschlüsseln des Paßwortes (mit dem auf der Karte gespeicherten Schlüssel).
- Speichern des verschlüsselten Paßwortes auf der Chipkarte.
- Speichern der Benutzerpräferenzen auf der Karte.

Erstellen der Benutzerkennung: Die Benutzerkennung (*login*-Name und Paßwort) wird vom Systemverwalter eingerichtet. Hierfür wird ein systemspezifisches Programm verwendet. Zusätzlich personalisiert der Systemverwalter eine Chipkarte für jeden Benutzer. Benutzerdaten werden von einem JAVA Applet eingelesen:

1. Erzeugen der benötigten Dateien (siehe nächster Abschnitt),
2. Schreiben der Dateiinhalte,
3. Abschluß der Personalisierungsphase,
4. Verschlüsseln des Paßwortes mit dem öffentlichen Schlüssel und
5. Speichern des verschlüsselten Paßwortes in einer Datei.

Von der Anwendung benötigte Dienste:

Die Anwendung benötigt vier Dienste der Chipkarte:

- den Dateisystemdienst,
- den kryptographischen Dienst,
- den Dienst zur Verwaltung der Sicherheitscodes und
- den Personalisierungsdienst.

Diese Anwendung wird mit Hilfe von OPENCARD entwickelt, um zu verhindern, daß der Anwendungsprogrammierer direkt auf Betriebssystemebene auf das Kartenschreiblesegerät zugreifen muß. Das OPENCARD-System stellt das nötige Gerüst zur Verfügung.
Für die verwendete Karte werden die Diensttreiber für das Dateisystem, die kryptographischen Funktionen und die Verwaltung der Sicherheitscodes benötigt. Eine vom OPENCARD-System standardisierte Schnittstelle ist bisher nur für den Dateisystemdienst vorhanden: Die anderen Dienste werden über selbst entworfene Schnittstellen angesprochen.

4.4 Treiber für die DXPLUS Chipkarte

Die Klassen, die zur Benutzung der DXPLUS-Karte verwendet werden, sind im Rahmen dieser Arbeit entwickelt und implementiert worden. Die Treiber für das PHILIPS PE122-210 Kartenschreiblesegerät und die Dienste der DXPLUS-Chipkarte sind in das OPENCARD-System integriert worden.

4.4.1 Integration von Chipkarten in das OPENCARD-System

Um Karten in das System zu integrieren, werden zunächst die Diensttreiber entwickelt. Eine konkrete Diensttreiberfabrik wird entwickelt und bei der Registrierung für Kartendienste angemeldet. Das OPENCARD-System instantiiert Diensttreiber durch die konkrete Diensttreiberfabrik für diese Chipkarte.
Der Anwendungsprogrammierer verwendet OPENCARD, um zu verhindern, daß die Anwendung auf Betriebssystemebene auf das Kartenschreiblesegerät zugreifen muß (vergleiche Abbildung 4.5 auf der nächsten Seite). Das OPENCARD-System stellt das nötige Gerüst zur Verfügung.
Durch die Verwendung von OPENCARD vereinfacht sich die Treiberentwicklung für Chipkarten, da ein Diensttreiber für das OPENCARD-System zu entwickeln ist und auf betriebssystem- und hardwarenahe Programmierung verzichtet werden kann. Ein Anwendungsprogammierer kann auf eine hochsprachliche Schnittstelle aufsetzen, die die gesamte Kommunikation mit der Chipkarte kapselt.

4.4.2 Dienste der DELARUE DXPLUS-Karte

Die für die DXPLUS-Chipkarte des Herstellers DELARUE CARD SYSTEMS entwickelten Diensttreiber werden betrachtet. Die DXPLUS ist eine Karte für hohe Sicherheitsanforderungen. Die DXPLUS kann den RSA-Algorithmus mit bis zu 1024 Bits Blocklänge ausführen. Das Dateisystem dieser Karte verfügt über eine Verzeichnisebene. In der Benutzungsphase können keine

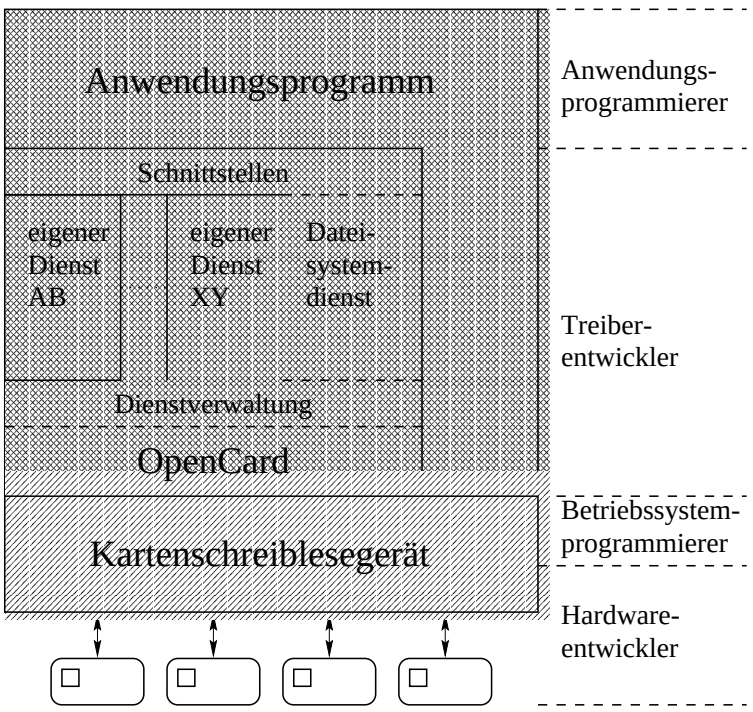

Abbildung 4.5: Integration von neuen Diensttreibern in das OPENCARD-System

Veränderungen mehr am Dateisystem vorgenommen werden. Alle Dateien, die in der Benutzungsphase verwendet werden sollen, müssen in der Personalisierungsphase erzeugt werden. Auf der DXPLUS Chipkarte sind verschiedene Klassen von Elementarfunktionen zu finden.

- **Dateisystemfunktionen:** Die Karte kann Dateien erzeugen, schreiben, lesen und sperren. Es ist möglich Dateiinformationen abzufragen. Die vom Dateisystem verwendeten Befehle sind `select`, `readBinary`, `updateBinary`, `createBinary`, `invalidate` und `getResponse`.

- **kryptographische Funktionen:** Die Karte kann den RSA-Algorithmus ausführen. Die verwendeten Befehle sind `computeRSA` und `getResponse`.

- **Verwaltungsfunktionen für Sicherheitscodes:** Sicherheitscodes (PINs) können geändert werden, blockierte PINs können entsperrt werden. Die verwendeten Befehle sind: `verifyPIN`, `changePIN`, `unblockPIN` und `getResponse`.

- **Funktionen zur Personalisierung:** Die verwendeten Befehle sind: `submitKey` und `getResponse`.

Um eine Karte zu personalisieren, werden Chipkarten verwendet, die über eine Funktion verfügen, die es ermöglicht, die individuellen Personalisierungsschlüssel für jede Karte zu berechnen (vergleiche Abschnitt 2.1.5 auf Seite 21).

4.4.3 Diensttreiber der DXPLUS Chipkarte

Dateisystemdienst

Der Zugriff auf das Dateisystem der DXPLUS-Karte erfolgt über den Dateisystemtreiber, der die Funktionen des Dienstes verfügbar macht. Das Dateisystem einer Chipkarte muß, bevor auf Dateien zugegriffen werden kann, für das System verfügbar gemacht werden, da die Chipkarte ein aus dem Schreiblesegerät entnehmbares, mobiles Medium ist. Der Dateisystemdienst, der für die DXPLUS-Karte entwickelt worden ist, ist an die OPENCARD-Dateisystemschnittstelle angepaßt worden.
In Computersystemen erfolgt der Zugriff auf Dateisysteme über Dateien und nicht über Funktionen eines Dienstes. Daher wird vom Dateisystemdienst die Möglichkeit zur Verfügung gestellt, über Dateiobjekte auf Kartendateien zuzugreifen: Es können Chipkartendatei-Objekte vom Dienst erzeugt werden, die als Dateien angesprochen werden (siehe Abbildung 4.6). Diese Dateien verwenden den Dienst, um auf ihnen definierte Operationen auszuführen.

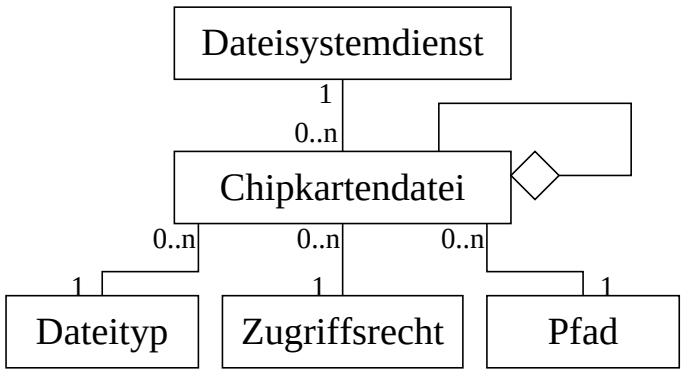

Abbildung 4.6: Zugriff auf den Dateisystemdienst durch Dateien

Der Zugriff auf Chipkartendateien ist eng an die JAVA-Schnittstelle für Dateien angelehnt. Dateien werden in JAVA über die Klasse `java.io.File` angesprochen. Für den Anwendungsprogrammierer wird auf diese Weise eine bekannte Schnittstelle zur Verfügung gestellt. Die von der Chipkartendateiklasse verstandenen Nachrichten sind analog zu den in JAVA gebräuchlichen Dateimethoden modelliert. Eine Datei ist über einen Pfad erreichbar. Falls es sich um ein Verzeichnis handelt, kann eine Liste der enthaltenen Dateien angefragt werden. Die Dateizugriffsrechte können überprüft werden, Lesen und Schreiben ist über *streams* realisiert.
Der Dateisystemdienst der DXPLUS-Karte entspricht nicht vollständig der ISO-Dateisystemnorm, sondern verwendet eine Untermenge der standardisierten Befehle. Nicht unterstützte Funktionen sind so implementiert, daß Fehlermeldungen wie in Abschnitt 3.5 auf Seite 46 beschrieben, generiert werden.
Das Dateisystem der DXPLUS-Chipkarte ist nicht dazu in der Lage, die laut ISO-Norm möglichen 255 Bytes hintereinander zu schreiben. Der Diensttreiber für den Dateisystemdienst ermöglicht das Schreiben von größeren Datenpaketen durch wiederholten Aufruf der entsprechenden Funktion der Karte.

Chipkartendateien Um Zugriff auf eine Chipkartendatei zu bekommen, wird ein Objekt der Klasse `CardFile` instantiiert. Tabelle 4.1 auf der nächsten Seite zeigt die Methoden der

Dateiklasse.

Methode	Kurzbeschreibung
`create`	Erzeugen einer Kartendatei
`delete`	Löschen einer Kartendatei
`exists`	Existiert eine Datei mit diesem Namen?
`fileInfo`	Auslesen der Dateiinformationen im Dateikopf
`fileSystemCardService`	zugehöriger Dateisystemdiensttreiber
`fileType`	Dateityp der Datei (siehe Dateityp)
`isDF`	Ist diese Datei eine Verzeichnisdatei (*dedicated file*)?
`isEF`	Ist diese Datei eine normale Datei (*elementary file)?*
`length`	aktuelle Größe der Datei
`list`	Liste aller Dateien im Verzeichnis (bei Verzeichnissen)
`listAsString`	wie `list` als Liste von Dateinamen
`mode`	Zugriffsrechte der Datei lesen/setzen
`name`	Dateiname
`parent`	Verzeichnis, in dem sich diese Datei befindet
`path`	Pfad zu dieser Datei
`readable`	ist die Datei lesbar?
`writable`	ist die Datei schreibbar?

Tabelle 4.1: Methoden der Dateiklasse `CardFile`

Dateityp Alle in der ISO-Norm definierten Dateitypen finden eine Repräsentation auf Ebene des Dateisystemdienstes. Die Klasse `CardFileType` stellt die Dateitypen zur Verfügung (vergleiche Tabelle 4.2 und Abschnitt 2.1.1 auf Seite 13).

Konstante	Beschriebener Dateityp
`CYCLIC_FIXED`	Datei mit zyklischer Struktur und fester Feldlänge
`DEDICATEDFILE`	Verzeichnisdatei
`DIRECTORY`	Verzeichnisdatei
`LINEAR_FIXED`	Datei mit fester Feldlänge
`LINEAR_VARIABLE`	Datei mit variabler Feldlänge
`TRANSPARENT`	Binärdatei

Tabelle 4.2: Dateitypen der Klasse `CardFileType`

Zugriffsrechte Zugriffsrechte für Dateien werden für verschiedene, auf den Dateien definierte Operationen vergeben. In der ISO-Norm definierte Operationen sind als Konstanten in der Klasse `CardAccess` definiert (siehe Tabelle 4.3 auf der nächsten Seite).
Die Zugriffsrechte für jede dieser Operationen sind in Tabelle 4.4 auf der nächsten Seite aufgeführt und in der Klasse`CardAccess` deklariert. Sie werden mit der Methode `queryPermission` ausgelesen.

Operation	Beschreibung
`CREATE_ACCESS`	Daten an Datei anhängen
`INVALIDATE_ACCESS`	Datei sperren
`READ_ACCESS`	Bytes aus Datei lesen
`REHABILITATE_ACCESS`	Datei entsperren
`UPDATE_ACCESS`	Daten in Datei ändern

Tabelle 4.3: Mit Zugriffsrechten schützbare Dateioperationen

Zugriffsrecht	Beschreibung
`ALWAYS_AC`	immer ausführbar
`CHV_AC`	nur nach Eingabe einer PIN (*card holder verification*)
`NEVER_AC`	nie ausführbar

Tabelle 4.4: Dateizugriffsrechte

Pfad Chipkartendateien haben 2 Bytes lange Dateinamen. Um diese anzusprechen, brauchen sie nicht als Binärcodes eingegeben zu werden. Ein Chipkartendateiname besteht aus dem Präfix „:“ und einer vierstelligen Hexadezimalzahl. Der Hauptverzeichniseintrag wird in dieser Notation als `:3F00` dargestellt. Die Klasse `CardFilePath` enthält Methoden, um auf solche Pfadnamen zu erzeugen.
Um einen Pfad anzugeben, werden wie in anderen Dateisystemen Verzeichnisnamen mit Hilfe eines Pfadtrenners aneinandergehängt. `:3F00:0004` ist in dieser Notation die Datei mit dem „Namen” 0004_{hex} im Hauptverzeichnis.

Verschlüsselungsdienst

Die für den kryptographischen Dienst entwickelte Schnittstelle ist so flexibel gehalten, daß auch Karten, die einen symmetrischen kryptographischen Algorithmus verwenden, benutzt werden können. Es wird eine gemeinsame Basisklasse für alle kryptographischen Algorithmen mit Subklassen für symmetrische (z.B. DES) und asymmetrische Verfahren (z.B. RSA) verwendet (siehe Abbildung 4.7).
Der Programmierer unterscheidet bei der Verwendung der Algorithmen nicht, um was für einen Algorithmus es sich handelt, sondern nur bei Generierung und Import der Schlüssel.

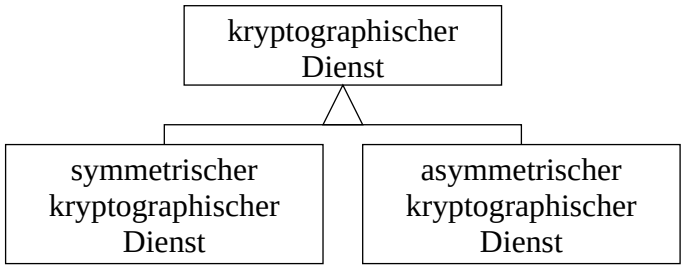

Abbildung 4.7: Der kryptographische Dienst

Die Superklasse enthält die Methoden, den kryptographischen Algorithmus zu verwenden. Die beiden Subklassen enthalten zusätzlich Methoden, um Schlüssel auf der Karte zu speichern.
Asymmetrische kryptographische Verfahren werden für verschiedene Aufgaben verwendet:

- **Verschlüsselung** und **Entschlüsselung** von Daten
- **digitales Signieren** von Daten
- **Authentifizieren** der Karte

Um diese Funktionen ausführen zu können, wird für die DeLaRue DXPLUS-Karte der kryptographische Diensttreiber `CryptoCardService` verwendet. Dieser Dienst macht den von der Karte berechenbaren RSA-Algorithmus für Benutzer verwendbar. Die Funktionsweise des Algorithmus wird in Abschnitt 2.1.6 auf Seite 22 erläutert.
Der Verschlüsselungsdienst stellt eine Methode zur Verfügung, die den RSA-Algorithmus der Karte mit den notwendigen Parametern ausführt und das Ergebnis an den Benutzer übermittelt.
Zur Verwendung des RSA-Algorithmus ist es erforderlich, den jeweils erforderlichen Schlüssel für diese Operation auf der Chipkarte in einer Datei abzulegen. Der Verschlüsselungsdienst überprüft, ob der verwendete Schlüssel im richtigen Format vorliegt, und schreibt diesen Schlüssel durch Verwendung des Dateisystemdienstes in eine Chipkartendatei. Bevor der Schlüssel geschrieben wird, muß überprüft werden, ob die Datei für den Schlüssel lesegeschützt ist, da geheime Schlüssel nicht lesbar sein dürfen.
Informationen über Schlüssellängen können nur bei Schlüsseln, die in lesbaren Dateien abgelegt sind, ausgelesen werden. Für geheime Schlüssel ist daher keine Information über die Schlüssellänge vorhanden.

PIN-Verwaltungsdienst

Die Ausführung von Funktionen auf Chipkarten ist durch persönliche Identifikationsnummern schützbar. Chipkarten erlauben hierfür bis zu 64 Bits lange Werte. Verwendet werden jedoch normalerweise nur vier Ziffern, da dies ein schon bei Magnetkarten gebräuchlicher Standard ist. Es ist üblich, eine PIN nach drei falschen Eingabeversuchen zu sperren: Dies wird selbstständig vom COS der Karte durchgeführt. Um die PIN wieder zu entsperren, ist ein weiterer 64 Bits langer Code erforderlich.
Der PIN-Verwaltungsdienst ermöglicht die Überprüfung einer PIN durch die Chipkarte und das Entsperren der PIN, falls sie gesperrt ist. Außerdem ist es möglich, die PIN zu verändern, sofern die Karte dies zuläßt. Da es Karten mit einem komplexen PIN-Verwaltungssystem gibt, die über mehrere PINs verfügen, ist vorgesehen, daß alle Funktionen mit verschiedenen PINs arbeiten, obwohl die DeLaRue DXPLUS-Chipkarte dies nicht unterstützt.
Auf der DeLaRue DXPLUS-Karte können folgende Funktionen mit einer PIN geschützt werden: `computeRSA`, `readBinary`, `updateBinary`, `createBinary` und `invalidate`.

Personalisierungsdienste

Generieren des Schlüssels Bevor eine Chipkarte in die Hände des Endkunden/Benutzers kommt, muß sie personalisiert werden (siehe Abschnitt 2.1.5 auf Seite 21). Hierfür wird

von einer zweiten Karte, die separat zu jeweils einem Paket von Karten verschickt wird, der Schlüssel zum Freischalten der Karte generiert.
Über diesen Dienst verfügen nur Karten des Typs *batch card*. Um die andere Karte freizuschalten, wird ein Dienst zur Berechung des kartenindividuellen Personalisierungsschlüssels verwendet, der den Dateisystemdienst benutzt.
Der Diensttreiber verfügt nur über zwei Methoden:

- Überprüfen, ob die vorliegende Chipkarte zur *batch card* paßt.
- Generieren des Schlüssels zum Freischalten der Karte.

Freischalten einer Karte und Übergang in den benutzbaren Zustand Dieser Dienst macht durch Personalisierungsschlüssel geschützte Karten schreibbar und überführt zum Abschluß der Personalisierungsphase die Karte irreversibel in den Benutzungszustand (vergleiche auch Abschnitt 2.1.5 auf Seite 20).
Um eine Karte freizuschalten, wird eine Kartenfunktion verwendet, die nur in der Personalisierungsphase verfügbar ist.
Die Karte wird nach Abschluß der Personalisierung in die Benutzungsphase überführt, indem mit einem Kartenbefehl eine Sperre geschrieben wird.

4.5 Benötigte Dateien

Die Karte speichert nur die zur Anmeldung beim Rechnersystem und zum Start der vom Benutzer bevorzugten Programme erforderlichen Daten. Benötigt werden (vergliche Abbildung 4.8):

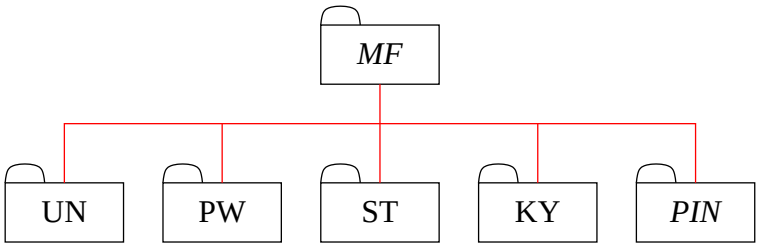

Abbildung 4.8: Auf den Karten benötigte Dateien

- Eine Datei, in der die Benutzerkennung gespeichert ist:

  Name: „UN“

  Länge: 16 Bytes

  Lesbar: Immer

  Schreibbar nach Eingabe einer PIN

- Eine Datei, in der das verschlüsselte Paßwort gespeichert ist:

  Name: „PW“

Länge: 8 Bytes

Lesbar: Immer

Schreibbar nach Eingabe einer PIN

- Eine Datei, in der die Namen der vom Benutzer bevorzugten Programme gespeichert werden:

  Name: „ST“

  Länge: minimal 1 KByte

  Lesbar: Immer

  Schreibbar Nach Eingabe einer PIN

- Eine Datei, die den öffentlichen Schlüssel zur Verschlüsselung des Paßwortes enthält:

  Name: „KY“

  Länge und Zugriffsrechte bestimmt das COS der Karte

- Eine Datei, die die PIN enthält:

  Name, Länge und Zugriffsrechte bestimmt das COS der Karte

Auf der DeLaRue DXPLUS-Karte können nach der Personalisierung keine weiteren Dateien erzeugt werden. Nicht belegter Speicher der Karte geht verloren. Für die Datei, die die Namen der vom Benutzer bevorzugten Programme enthält, wird der gesamte nach der Erzeugung aller anderen Dateien noch verfügbare Speicher verwendet. Alle Dateien werden direkt im Hauptverzeichnis angelegt, da die Karte keine Unterverzeichnisse erzeugen kann.

Kapitel 5

Diskussion und Ausblick

5.1 Diskussion der Ergebnisse

In dieser Arbeit wird ein objektorientiertes Modell für die Benutzung von Chipkarten entworfen und implementiert. Das Modell wird mit Hilfe der *Unified Modeling Language* unter Verwendung von objektorientierten Entwurfsmustern beschrieben. Szenarien der Verwendung von Chipkarten werden auf ihre Anwendungsfälle hin analysiert. Die Analyse orientiert sich an Anwendungen, die Chipkarten zur Zugriffskontrolle zur Speicherung von Benutzerdaten verwendet.

Objektorientierte Modellierung

Anhand einer Anwendungsfallanalyse wird das Modell der Dienstleistungen für das aktive Medium Chipkarte entwickelt. Dienstleistungen sind ein den Merkmalen der Chipkarte angemessenes Modell; es wird berücksichtigt, daß die Karte Funktionen ausführen kann. Objektorientierte Treiber für die Dienste kapseln die Funktionen der Karte so, daß die auf Protokollebene verwendeten Bytecodes für den Anwendungsprogrammierer nicht mehr sichtbar sind. Vorteile der objektorientierten Modellierung und Implementierung sind:

- Durch die Kapselung der Kartenbefehle in Treiberklassen sind für den Benutzer herstellerspezifische Merkmale der Karte nicht sichtbar; intern werden sie zur Leistungssteigerung der Treiber ausgenutzt.
- Die Verwendung der Treiberklassen als Diensterbringer ermöglicht einen Zugriff auf Chipkartenhardware durch flexibel einsetzbare, hochsprachliche Treiber.
- Identische Schnittstellen für verschiedene Chipkarten ermöglichen den Wechsel auf andere Karten mit minimalem Aufwand.

Verwendung von Chipkarten in persistenen und verteilten Systemen

Chipkarten werden in verteilten Systemen, die auf persistente Hintergrundsysteme zugreifen, verwendet. Das vorgestellte Modell ermöglicht den Aufbau verteilter Systeme durch die Verwendung von JAVA. Der Zugriff auf persistente Hintergrundsysteme ist ebenfalls von JAVA aus möglich. Diese Systeme authentifizieren Benutzer mit Hilfe von Daten und Funktion auf

Chipkarten. Auf der Karte abgelegte Daten und Zustände der Sicherheitscodes sind persistent. Die Karte kann damit das Sicherheitssystem des Hintergrundsystems ergänzen.

Plattformunabhängigkeit

Ein vollständig plattformunabhängiger Zugriff auf Chipkarten und Kartenschreiblesegeräte kann durch Verwendung der vorgestellten Routinen nicht ermöglicht werden, da es sich um Hardware handelt. Das entworfene objektorientierte Modell berücksichtigt daher, daß ein Zugriff auf Geräte nicht hardwareunabhängig durchgeführt werden kann. Durch den Einsatz objektorientierter Entwurfsmuster wird eine Lösung gefunden, Treiber für Kartenschreiblesegeräte flexibel für verschiedene Plattformen zu instantiieren. Die Treiber werden für verschiedene Chipkarten auf die gleiche Weise dynamisch instantiiert. Der Programmcode ist so entwickelt, daß die hardwareabhängigen Klassen klein sind. Wenig Code muß angepaßt werden, um Geräte und Chipkarten zu integrieren. Die Anpassung der Implementierung für Kartenschreiblesegeräte auf verschiedenen Plattformen und für verschiedende Chipkarten wird damit vereinfacht.

Persistenzaspekt der Chipkarte

Die Persistenz der Kartenzustände wird in den entwickelten Diensttreibern berücksichtigt. Zugriff auf eine Chipkarte wird nicht versucht, wenn diese gesperrt ist.

Mobilität der Chipkarte

Chipkarten sind physikalisch mobile aktive Datenträger. Sie können an ein System angekoppelt und aus diesem entnommen werden. Das entworfene System berücksichtigt die Mobilität durch eine Ereignisverwaltung. Objekte werden benachrichtigt, wenn Chipkarten in ein Schreiblesegerät eingelegt oder aus diesem entnommen werden. Wird eine Karte während eines Zugriffes entnommen, wird dies durch eine Ausnahme signalisiert.

Berücksichtigung von Normen

Internationale Normen für Chipkarten werden bei der Entwicklung der Treiber verwendet. Die einzige standardisierte Gruppe von Chipkartenfunktionen sind die Dateisystemfunktionen. Der Dateisystemdienst ist so modelliert, daß normgerechte Karten mit minimalen Anpassungen integrierbar sind. Für den Zugriff auf Chipkarten und Schreiblesegeräte von Programmiersprachen aus existieren keine verbindlichen Richtlinien. Die Entwicklung des OPENCARD-Frameworks ermöglicht es, ein von Chipkartenherstellern, Schreiblesegeräteherstellern und Softwarehäusern unterstütztes Referenzmodell zu verwenden.

Unabhängigkeit von der verwendeten Chipkarte

Eine Unabhängigkeit der implementierten Treiber vom Chipkartentyp konnte nicht vollständig erreicht werden. Für viele Chipkartenfunktionen existieren keine verbindlichen Vorschriften für die zu verwendenden Bytecodes. Durch Verwendung einheitlicher Schnittstellen für die Diensttreiber kann für die Anwendung der Zugriff auf verschiedene Karten trotzdem identisch stattfinden.

Beispielanwendung

Die entwickelte Anwendung „Anmeldung bei einem Rechnersystem mit Chipkarte“ hat die Funktionsfähigkeit der beschriebenen Konzepte nachgewiesen. Es wird gezeigt, daß Chipkarten sich als sichere Speichermedien für Daten zur Authentifizierung und Autorisierung von Benutzern eignen. Chipkarten ermöglichen es durch das Ausführen eines kryptographischen Algorithmus, Daten zu verschlüsseln. Auf keinem anderen elektronischen Medium gleicher Größe und Mobilität existieren ähnliche Möglichkeiten.

5.2 Weiterführende Arbeiten

In dieser Arbeit hat sich gezeigt, daß plattformunabhängiger Zugriff auf Chipkarten auch durch einen flexiblen objektorientierten Entwurf nur teilweise möglich ist. Die Einschränkungen ergeben sich daraus, daß Chipkarten und Kartenschreiblesegeräte Hardwarekomponenten sind, die nur teilweise standardisiert sind.

Schreiblesegeräte für Chipkarten

Treiber für Chipkartenschreiblesegeräte greifen auf Hardware zu und können im allgemeinen nicht plattformunabhängig entworfen werden. Treiber für Geräte mit serieller Schnittstelle können durch Verwendung der plattformunabhängigen Unterstützung für serielle Schnittstellen für alle Plattformen identisch entwickelt werden. Diese Erweiterungen sind für neuere Versionen des JAVA-JDK verfügbar.

Treiber für Kartendienste

Unabhängigkeit der Kartendiensttreiber von der verwendeten Chipkarte kann durch die Einführung definierter Schnittstellen erreicht werden. Es müssen Treiber für die Dienste jeder Karte implementiert werden, die diesen Schnittstellen entsprechen. Die Standardisierung der Schnittstellen muß zentral und verbindlich für alle Benutzer von Chipkarten durchgeführt werden.
Benötigte Treiber für Chipkartendienste können durch die Verwendung von JAVA automatisch für die jeweils verwendete Karte geladen werden. Das Nachladen kann aus Dateien oder über die Netzwerkschnittstelle von JAVA gelöst werden. Eine weitere Möglichkeit ist das Laden von Diensttreibern für jede Karte aus Chipkartendateien. Die Diensttreiber müssen im Dateisystem der Karte vom Kartenhersteller abgelegt werden. Hierfür ist ein Zugriff auf das Dateisystem der Chipkarte durch den Dateisystemdiensttreiber erforderlich. Dateisystemfunktionen von Chipkarten sind durch eine internationale Norm festgelegt. Das Laden von Treibern aus der Chipkarte ist bei normgerechten Karten über einen für alle Karten identischen Dateisystemdienst möglich. Einziges Problem ist die Speichergröße der Karten. Es ist momentan nicht möglich, alle zu einer Karte gehörenden Diensttreiber in deren Speicher abzulegen. Die Entwicklung der Speicher-Hardware schreitet jedoch schnell voran.
Treiber für Dienste von Chipkarten können noch nicht dynamisch generiert werden, da es keine Möglichkeit gibt, von einer Karte zu erfragen, welche Funktionen sie anbietet und wie diese Funktionen arbeiten. Automatische Abfrage von diesen für jede Karte spezifischen Daten ist jedoch möglich. Verschiedene Lösungen dafür bieten sich an:

- Auf Chipkarten ohne Programmausführungsmöglichkeit kann eine Tabelle der verfügbaren Funktionen und Parameter im ROM abgelegt werden, wenn das COS der Karte geschrieben wird.
- Auf programmierbaren Chipkarten gibt es die Möglichkeit, Beschreibungen der verwendbaren Befehle und Parameter mit einem Chipkartenprogramm an das Kartenschreiblesegerät zu übergeben.

Aus solchen Beschreibungen ist es möglich, automatisch Diensttreiber zu generieren. Die technischen Möglichkeiten für ein solches Vorgehen sind vorhanden. Bisher existieren keine Normen für die Beschreibung von Chipkartenfunktionen; bevor sich solche Beschreibungen herstellerübergreifend einsetzen lassen, müssen sie standardisiert werden.
Chipkarten mit Programmausführungsmöglichkeit bieten Dienste an, die vom geladenen Programm definiert werden. Wenn zur Programmierung der Chipkarte JAVA verwendet wird, ist es möglich, aus dem Code des Chipkartenprogrammes einen Teil des Diensttreibers automatisch zu generieren. Eine entsprechende Funktion muß in die Entwicklungsumgebung für Chipkartenprogramme eingebunden werden.
Nach der zu erwartenden Standardisierung der Beschreibungsformate kann ein Dienst zum Generieren von Treibern in das entworfene Modell eingefügt werden. Dies ist durch die flexible Struktur des entworfenen Modells einfach möglich.

Ausblick

Bisher ist es nicht möglich, mehrere Prozesse parallel auf einer Chipkarte auszuführen. Auch auf JAVA-Karten kann nur ein Programm zur Zeit aktiv sein. Programme können in den Kartenspeicher geladen und aus dem Speicher gelöscht werden. Der interne Status eines laufenden Programmes ist persistent, aber nicht zugänglich. Mit Chipkarten ist durch Migration von Kartenprogrammen durch ein Kartenschreiblesegerät auf eine andere Karte keine Mobilität von Prozessen realisierbar. Chipkarten, die Programme ausführen können, bieten die Möglichkeit, physikalisch mobile persistente Prozesse zu verwenden, die durch das Sicherheitssystem der Karte geschützt sind. Diese Fähigkeiten sind noch weitgehend ungenutzt, da die Speichergröße und Programmausführungsgeschwindigkeit von Chipkarten beschränkt sind.

Bei der zu erwartenden Leistungssteigerung der Chipkartentechnologie und dem damit verbundenen Gewinn an Bedeutung, wird die Nachfrage für Modelle und Lösungsansätze zur Benutzung von Chipkarten steigen. Die in dieser Arbeit gefundenen Lösungen können als Basis weiterführender Projekte eingesetzt werden.

Literaturverzeichnis

Coulouris et al. 94: Coulouris, George, Dollimore, Jean, und Kindberg, Tim. *Distributed Systems, concepts and designs*. Addison Wesley, 2nd edition, 1994.

Dal 97: Dallas Semiconductor. *iButton technical manual*, 1997.

DeL 97: DeLaRue Products and Services, http://www.delarue.com/cards/projects/medea/index.html. *MASSC A 112*, 1997.

DeL 98a: DeLaRue Cartes et Systems SAS. *DX Smart Card Reference Manual*, Jan 1998.

DeL 98b: DeLaRue Cartes et Systems SAS. *DXPLUS Smart Card Reference Manual*, Mar 1998.

Eckel 98: Eckel, Bruce. *Thinking in Java*. Prentice Hall PTR, 1998.

Fowler 97: Fowler, Martin. *UML Distilled*. Addison Wesley, 1997.

Gamma et al. 95: Gamma, Erich, Helm, Richard, Johnson, Ralph, und Vlissades, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Herrmann, Husemann 98: Herrmann, Reto und Husemann, Dirk. *OpenCard Framework 1.0 White Paper*. Technical report, IBM Research Division, http://www.opencard.org/docs/whitepaper, May 1998.

IBM 98: IBM United Kingdom Ltd. *IBM Multifunction SmartCard with public Key*, 1998.

ISO 87: ISO/IEC. *International Standard ISO/IEC 7816-1*, 1987.

ISO 88: ISO/IEC. *International Standard ISO/IEC 7816-2*, 1988.

ISO 89: ISO/IEC. *International Standard ISO/IEC 7816-3*, 1989/92/93.

ISO 94: ISO/IEC. *International Standard ISO/IEC 7816-4*, 1994.

Johannisson 97: Johannisson, Nico. *Eine Umgebung für mobile Agenten: Agentenbasierte verteilte Datenbanken am Beispiel der Kopplung autonomer „Internet Web Site Profiler“*. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, April 1997.

Jugel, Meißner 97: Jugel, Matthias L. und Meißner, Marcus. *The Java(tm) Telnet Applet*. Technical report, GMD, http://www.first.gmd.de/persons/leo/java/Telnet/, 1997.

Mathiske 96: Mathiske, Bernd. *Mobilität in persistenten Objektsystemen*. Dissertation, Fachbereich Informatik, Universität Hamburg, Mai 1996.

Matthes 93: Matthes, Florian. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer Verlag, 1993.

Matthias Kaiserswerth 98: Matthias Kaiserswerth, Joachim Prosegga. *Java auf Chipkarten*. Informatik Spektrum, Februar 1998, Nr. 21, S. 27–28.

Oesterreich 97: Oesterreich, Bernd. *Objektorientierte Softwareentwicklung mit der Unified Modeling Language*. Oldenbourg, 1997.

Ope 98: OpenCard Group, http://www. opencard.org/docs/packages.html. *OpenCard Framework Java Documentation (in javadoc format)*, May 1998.

OpenCard 98: OpenCard. *OpenCard Web-Homepage*. Technical report, OpenCard Group, http://www.opencard.org, 1998.

Platzer 98: Platzer, Wolfgang. *IAIK Java Security*. Technical report, GMD, http://jcewww.iaik.tu-graz.ac.at/, 1998.

Sun 97: Sun. *The java.smartcard API*. Technical report, Sun Microsystems, http://www. java.sun.com, 1997.

Sun 98a: Sun. *Java Card 2.0 Application Programming Interfaces*. Technical report, Sun Microsystems, http://java.sun.com/products/javacard/index.html, 1998.

Sun 98b: Sun. *Java Card 2.0 Language Subset and Virtual Machine Specification*. Technical report, Sun Microsystems, http://java.sun.com/products/javacard/index.html, 1998.

Sun 98c: Sun. *JavaStation(TM) NC*. Technical report, Sun Microsystems, http://www.sun.com/javasystems/javastation/, 1998.

Sun 98d: Sun. *The JDBC Database Access API*. Technical report, SUN Microsystems, http://www.java.sun.com/products/jdbc, 1998.

Tanenbaum 92: Tanenbaum, Andrew S. *Modern Operating Systems*. Prentice Hall inc., 1992.

Wolfgang Rankl 96: Wolfgang Rankl, Wolfgang Effing. *Handbuch der Chipkarten*. Carl Hanser, München, Wien, 1996. 2. völlig überarbeitete und erweiterte Auflage.

Anhang A

Benutzung von OPENCARD

A.1 Beispiele für die Programmierung

A.1.1 Beispiel ohne Ereignissteuerung

```
class OpenCardTest {
public static void main(String[] argv) {
  // Start von OpenCard und laden des Dateisystemdienst´
  FileSystemCardService fileSystem = null;
  CardRequest cr = null;
  CardFile root = null, idFile = null;
  ...
  SmartCard.start();
  ...
  SmartCard card = SmartCard.waitForCard (cr);
  ...
  fileSystem =(FileSystemCardService)
         card.getCardService(FileSystemCardService.class,true);
  root = fileSystem.mount(CardFileOpenMode.BLOCKING);
  idFile = new CardFile(root,":0004");
  ...
  SmartCard.shutdown();
 }
}
```

Das Codefragment zeigt, wie das OPENCARD *Framework* gestartet wird und den Dateisystemdienst einer eingelegten Chipkarte anfordert.

1. Das System wird mit `SmartCard.start()` gestartet und Treiber für angeschlossene Schreiblesegeräte, die über die Konfigurationsdatei identifiziert werden, werden instantiiert.

2. Die `waitForCard()` Methode instantiiert ein Objekt der Klasse `SmartCard`, sobald eine Chipkarte in ein angeschlossenes Schreiblesegerät eingelegt wird.

3. Um einen Diensttreiber zu erzeugen, wird `getCardService()` verwendet. In diesem Beispiel wird ein Treiber für den Dateisystemdienst der Karte nachgefragt.

4. Das Dateisystem der Karte muß zunächst verfügbar gemacht werden: Die `mount()` Methode wird aufgerufen.

5. Jetzt können Dateien instantiiert werden die dann, je nach Zugriffsrechten, gelesen und geschrieben werden können.

A.1.2 Beispiel mit Ereignissteuerung

Um von Kartenereignissen benachrichtigt werdn zu können, muß ein Programm die Schnittstelle `CTListener` implementieren:

```
public class MainFrame implements CTListener {
   SmartCard card = null;
   FileSystemCardService fileSystem = null;

   public void cardInserted(CardTerminalEvent ctEvent) {
    System.out.println("CARD INSERTED");
    // first get the SmartCard Object for this event
    card = SmartCard.getSmartCard (ctEvent);
    fileSystem =(FileSystemCardService)
              card.getCardService(FileSystemCardService.class,true);
    ...
   }

   public synchronized void cardRemoved(CardTerminalEvent ctEvent) {
    System.out.println("CARD REMOVED");
    card = null;
    fileSystem = null;
    ...
   }

  public static void main(String[] argv) {
  ...
  SmartCard.start();
  // register this program for receving CTListener Events
  CardTerminalRegistry.getRegistry().addCTListener(this);
  ...
  }
}
```

1. Zunächst wird die `main()` Methode der Klasse ausgeführt

   - Das System wird wie im vorigen Beispiel mit `SmartCard.start()` gestartet.
   - Die Klasse, die die Schnittstelle `CTListener` implementiert, wird als Ereignisempfänger registriert.

2. Wenn eine Karte in ein angeschlossenes Schreiblesegerät eingelegt wird, wird die `cardInserted()` Methode aufgerufen:

- ,,CARD INSERTED" wird auf dem Bildschirm ausgegeben
- Ein `SmartCard` Objekt instantiiert
- Der Dateisystemdiensttreiber der Karte wird angefordert

3. Wird die Karte wieder enfernt, ruft die Ereignisverwaltung die `cardRemoved()` Methode auf:

   - ,,CARD REMOVED" wird auf dem Bildschirm ausgegeben
   - Die vorher instantiierten Variablen werden zurückgesetzt

A.2 Aufbau der Konfigurationsdatei

```
OpenCard.services =  \
       com.delarue.opencard.service.DXCardServiceFactory \
       com.ibm.opencard.service.MFCCardServiceFactory

OpenCard.terminals = \
    com.philips.opencard.terminal.PE122.PhilipsCardTerminalFactory\
|Terminal1|PhilipsPE122|CR1
```

Die beiden Stichworte `OpenCard.services =` und `OpenCard.terminals =` stehen für im System zu registrierende konkrete Fabriken: Diensttreiberfabriken werden als `services` registriert, die darauf folgende Zeichenkette ist der vollständige Klassenname der konkreten Fabrik (hier: Paket `com.delarue.opencard.service`, Klassenname `DXCardServiceFactory`.
Bei den Treibern für Kartenschreiblesegeräte wird genauso verfahren, die hier registrierte Fabrik befindet sich im Paket `com.philips.opencard.terminal.PE122` und hat den Namen `PhilipsCardTerminalFactory`. Die drei durch `|` abgetrennten Zeichenketten sind Parameter, die für die Instantiierung der angeschlossenen Schreiblesegeräte benötigt werden: Der erste Parameter ist ein frei wählbarer logischer Name für ein Schreiblesegerät, der zweite gibt den Typ des Gerätes an, unter dem die konkrete Fabrik es kennt und der dritte ist ein Parameter, der frei nutzbar ist und in diesem Fall für den Zugriff auf den Windows NT spezifischen Treiber für das Gerät verwendet wird.

Anhang B

Implementierte Klassen und Treiber

Beschreibung der Klassen mit dem Dateinamen, dem Klassennamen (inklusive Paketnamen) und einer kurzen Beschreibung der Aufgaben dieser Klasse. Die vollständige Dokumentation findet sich den den `javadoc`-Dateien zu den Klassen. Ein Klassendiagramm aller implementierten Klassen findet sich in Abbildung B.1.

B.1 Kartendiensterzeugung

Klasse: `com.delarue.opencard.services.DXCardServiceFactory`
Superklasse: `CardServiceFactory`

Datei: `DXCardServiceFactory.java`

Beschreibung: Erzeugt Treiber für Kartendienste der DeLaRue DX und DXPLUS Karten sowie der *batch*-Karten.

Klasse: `com.delarue.opencard.services.DXConstants und DXResponseAPDUCodes`

Datei: `DXConstants.java, DXResponseAPDUCodes.java`

Beschreibung: Konstantendefinitionen: Die für alle Modelle aus der DX Chipkartenserie verwendeten Befehlscodes (INS-Codes) und die entsprechenden Antwortcodes

B.1.1 Dateisystemdienst

Klasse: `com.delarue.opencard.iso.fs.DXAccess`

Datei: `DXAccess.java`
Superklasse:`CardAccess`

Beschreibung: Zugriffsrechte für eine Datei können über Methoden dieser Klasse ausgelesen werden

Klasse: `com.delarue.opencard.iso.fs.DXBasicCardFileSystem`
Superklasse:`CardService`

Datei: `DXBasicFileSystemCardService.java`

Beschreibung: Die gemeinsame abstrakte Basisklasse für alle Dateisystemtreiber von Karten des DX-Typs.

Der Befehlssatz ist bei allen Modellen gleich, Unterschiede sind nur in den Parametern der Befehle zu finden: Die DXPLUS liest und schreibt 132 Bytes mit einem Befehl, die DX Karte nur 64; die DXPLUS führt den RSA-Algorithmus mit bis zu 1024 Bit Blocklänge aus, die DX nur mit 512 Bit.

Klasse: `com.delarue.opencard.iso.fs.DXFileSystemCSImpl`

Datei: `DXFileSystemCSImpl.java`

Beschreibung: Kapselt die nicht objektorientierten Kartenfunktionen für das Dateisystem der DELARUE DX-Karte in einem objektorientierten Treiber.

Klasse: `com.delarue.opencard.iso.fs.DXCardFile`
Superklasse: `CardFile`

Datei: `DXCardFile.java`

Beschreibung: Repräsentiert eine Datei auf einer DX/DXPLUS Chipkarte

Klasse: `com.delarue.opencard.iso.fs.DXCardFileInfo`
Superklasse: `CardFileInfo`

Datei: `DXCardFileInfo.java`

Beschreibung: Kapselt den Dateiinformationsblock einer DX/DXPLUS Chipkarte

Klasse: `com.delarue.opencard.iso.fs.DXPLUSFileSystemCSImpl`
Superklasse: `DXFileSystemCSImpl`

Datei: `DXPLUSFileSystemCSImpl.java`

Beschreibung: Dateisystemtreiber für das Dateisystem der DXPLUS-Karte

Klasse: `com.delarue.opencard.iso.fs.DXCardFileInfoCache`

Datei: `DXCardFileInfoCache.java`

Beschreibung: Speichert häufig verwendete Chipkartenbefehle zur Auswahl von Dateien und die Antworten der Karte

Klasse: `com.delarue.opencard.iso.fs.DXPLUSFileSystemCardService`
Superklasse: `DXBasicFileSystemCardService`

Datei: `DXPLUSFileSystemCardService.java`

Beschreibung: Erlaubt Zugriff auf das Dateisystem der DXPLUS Chipkarte

B.1.2 Verschlüsselungsdienst

Dieser Dienst macht den auf der Karte verwendeten RSA-Algorithmus verfügbar.

Schnittstellen

Klasse: «**interface**» `opencard.crypto.AsymCryptoCardService`
Superklasse: `CryptographicCardService`

Datei: `AsymCryptoCardService.java`

Beschreibung: Treiber für asymmetrische kryptographische Funktionen müssen diese Schnittstelle implementieren

Klasse: «**interface**» `opencard.crypto.CryptographicCardService`

Datei: `CryptographicCardService.java`

Beschreibung: Basisklasse für alle kryptographischen Dienste

Klasse: «**interface**»`com.delarue.opencard.SymCryptoCardService`
Superklasse: `CryptographicCardService`

Datei: `SymCryptoCardService.java`

Beschreibung: Diensttreiber für symmetrische kryptographische Algorithmen müssen diese Schnittstelle implementieren

Klassen

Klasse: `com.delarue.opencard.DXPLUSKeyInfo`

Datei: `DXPLUSKeyInfo.java`

Beschreibung: Instanzen dieser Klasse kapseln Informationen über kryptographische Schlüssel.

Klasse: `com.delarue.opencard.crypto.RSAKey`

Datei: `RSAKey.java`

Beschreibung: Schlüssel für den RSA-Algorithmus

Klasse: `com.delarue.opencard.cryptoDXPLUSCryptoCSImpl`

Datei: `DXPLUSCryptoCSImpl.java`

Beschreibung: Kapselt den Zugriff auf die nicht objektorientierten kryptographischen Kartenfunktionen der DXPLUS-Karte

Klasse: `com.delarue.opencard.crypto.DXPLUSCryptoCardService`
implementiert: `AsymCryptoCardService`

Datei: `DXPLUSCryptoCardService.java`

Beschreibung: Treiber für den kryptographischen Dienst der DXPLUS-Karte

B.1.3 Zugriffscode-Verwaltungsdienst

Klasse: `com.delarue.opencard.pin.DXPinCardService`

Datei: `DXPinCardService.java`
Superklasse:`CardService`

Beschreibung: Dienst, Sicherheitscodes zu ändern und um gesperrte Sicherheitcodes zu entsperren

B.1.4 Personalisierungsdienst

Klasse: `com.delarue.opencard.personalisation.DXPersonalisationCSImpl`

Datei: `DXPersonalisationCSImpl.java`

Beschreibung: Kapselt nicht objektorientierte Chipkartenfunktionen, die zur Personalisierung der Karte benötigt werden

Klasse: `com.delarue.opencard.personalisation.DXPersonalisationConstants`

Datei: `DXPersonalisationConstants.java`

Beschreibung: Vom Personalisierungsdienst verwendete Konstanten

Klasse: `com.delarue.opencard.personalisation.DXPersonalisationCardService`

Datei: `DXPersonalisationCardService.java`
Superklasse: `CardService`

Beschreibung: Treiber für den Personalisierungsdienst. Enthält Methoden, um den Personalisierungsschlüssel zu übermitteln und um die Personalisierungsphase irreversibel zu verlassen. Für DX und DXPLUS Chipkarte verwendbar

B.2 Schreiblesegerättreiber

Klasse: %

Datei: `PhilipsPE122Driver.c`

Beschreibung: Funktionen um mit dem Schreiblesegerät zu kommunizieren. WINDOWS NT 4.0 spezifisch

Klasse: `com.philips.opencard.terminal.pe122.PhilipsPE122Driver`

Datei: `PhilipsPE122Driver.java`

Beschreibung: Kapselt die nicht objektorientierten Funktionen der dynamischen Bibliothek in einer Klasse

Klasse: `com.philips.opencard.terminal.pe122.PhilipsPE122CardTerminal`
Superklasse: `CardTerminal`

Datei: `PhilipsPE122CardTerminal.java`

Beschreibung: Treiber für das Kartenschreiblesegerät, der vom OPENCARD-Framework verwendet wird

Klasse: `com.philips.opencard.terminal.pe122.PhilipsCardTerminalFactory`
implementiert: `CardTerminalFactory`

Datei: `PhilipsCardTerminalFactory.java`

Beschreibung: Erzeugt Treiberobjekte für Philips Kartenschreiblesegeräte

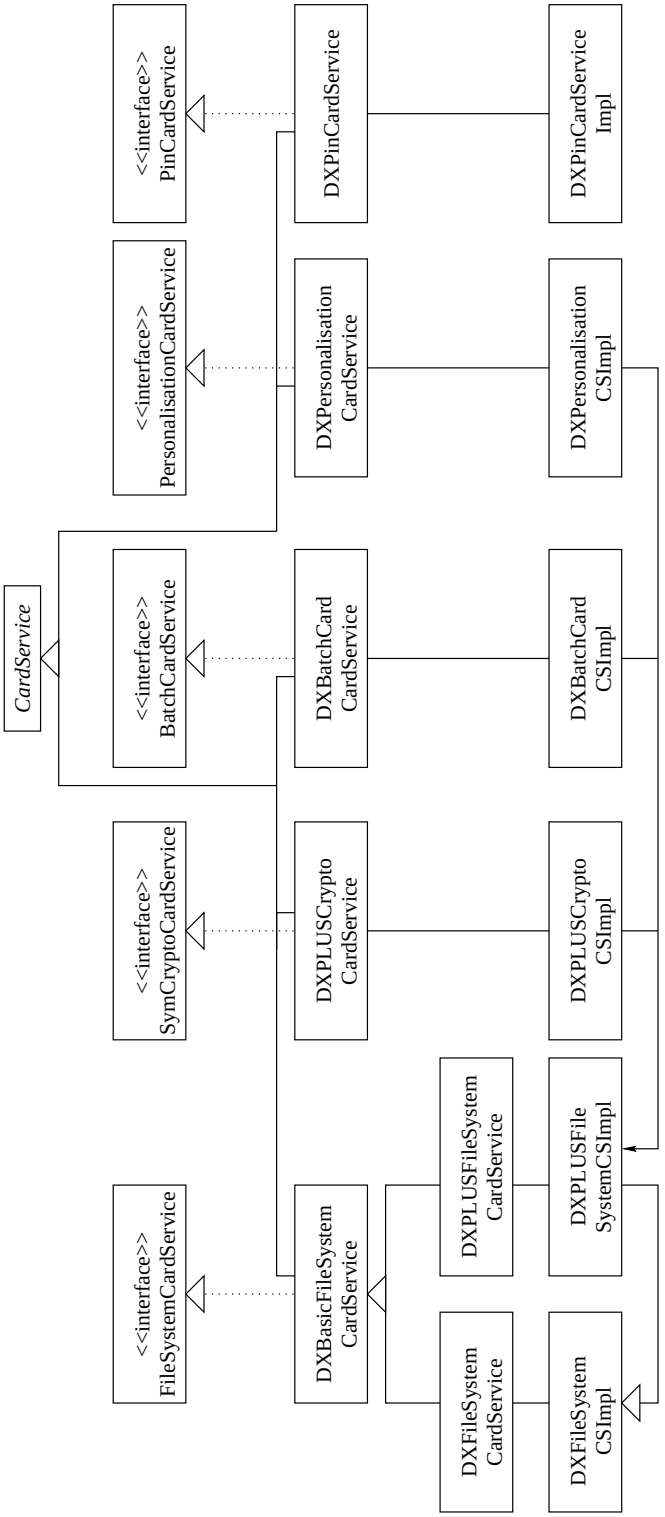

Abbildung B.1: Alle implementierten Klassen

Anhang C

Der RSA Algorithmus

Dieser Anhang erläutert kurz die dem RSA-Algorithmus zugrundeliegenden mathematischen Prinzipien.
Der öffentliche Schlüssel besteht aus zwei Komponenten, dem Exponenten und dem Modulus. Der Modulus n wird als das Produkt zweier großer Primzahlen p und q berechnet und seine Größe definiert die Blockgröße des Algorithmus. Der Modulus berechnet sich als:

$$n = p\,q$$

Der geheime Schlüssel d und der öffentliche Schlüssel e werden berechnet:

$$z := (p-1)(q-1)$$

$$e \in \{x \mid e < z \bigwedge ggT(e,z) = 1\}$$

(e ist kleiner als z und nicht Teiler von z, mehrere Zahlen erfüllen dies Kriterium, eine kann gewählt werden)

$$d \in \{x \mid (d\,e) \bmod z = 1$$

(Das Produkt aus geheimem und öffentlichem Schlüssel muß ein vielfaches von z sein. Die Verschlüsselung einer Nachricht x in eine verschlüsselte Nachricht x´ funktioniert folgendermaßen:

$$x' = x^e \bmod n,$$

Um x´ zu entschlüsseln wird der geheime Schlüssel verwendet, der einen größeren Exponenten und denselben Modulus verwendet. Die Entschlüsselung geschieht über:

$$x = x' = x^d \bmod n,$$

wobei d der geheime Schlüssel ist.

Anhang D

Abkürzungsverzeichnis

APDU application protocol data unit

ATR answer to reset

AT IBM PC-AT

COS Card Operating System

DES data encryption standard algorithm

DF dedicated file

EEPROM electrically erasable programmable read only memory

EF elementary file

ETU elementary time unit

IDEA intenational data encryption algorithm

JDBC Java Database Connectivity

JDK Java Development Kit

MASSC (Multi Application Secure SmartCard Projekt

MF masterfile

PCI peripheral component interface

PCMCIA Personal Computer Memory Card International Association

PIN personal identification number

RAM random access memory

REV remote evaluation

RMI remote method invocation

ROM read only memory

RPC remote procedure call

RSA Rivest Shamir Adleman algorithm

SQL structured query language

XT IBM PC-XT

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 31. Juli 1998

(Steffen Sperling)