Die JavaCard als Programmier- und Anwendungsplattform für verteilte Anwendungen[1]

Friedemann Mattern
(mattern@informatik.tu-darmstadt.de)
Stefan Fünfrocken
(fuenf@informatik.tu-darmstadt.de)

Technische Universtität Darmstadt
Fachbereich Informatik
Alexanderstraße 6
64283 Darmstadt
www.informatik.tu-darmstadt.de/VS

Marie-Luise Moschgath
(moschgath@ito.tu-darmstadt.de)

Technische Universität Darmstadt
ITO
Wilhelminenstraße 7
64283 Darmstadt
www.ito.tu-darmstadt.de

Kurzfassung

Chipkarten, die durch den integrierten Prozessor einfach zu nutzende Rechenkapazität an jedem beliebigen Ort bereitstellen, finden in immer mehr Bereichen Verwendung. Herkömmliche Smartcards besaßen aufgrund ihrer umständlichen Programmierung jedoch den Nachteil langer „Time-to-market"-Zeiten, zudem war es kaum möglich, mehr als eine Applikation auf der Karte ablaufen zu lassen. Dies hat sich mit der Verfügbarkeit neuer Smartcardtypen nun dramatisch geändert. So wurden in jüngster Zeit nicht nur neue Kartenbetriebsysteme entwickelt, die Mehrfachanwendungen auf der Karte ermöglichen, sondern auch Karten, die in einer Hochsprache (z.B. Java) programmiert werden können oder die es erlauben, Applikationen dynamisch auf die Karte nachzuladen.
Eine besondere Rolle spielen hier Java-basierte Smartcards. Mit der sogenannten JavaCard gelingt die Einbindung in offene bzw. netzbasierte Anwendungsszenarien besonders leicht, da Java-basierte Ausführungsumgebungen im Internet und in Intranets weit verbreitet sind. Sie stellt als Programmier- und Anwendungsplattform somit eine „portable" Komponente in einem verteilten Anwendungsverbund dar. Durch das Sicherheitskonzept von Java wird die Sicherheit der Kartenapplikationen zusätzlich erhöht.
Der Beitrag gibt einen Überblick zur JavaCard, skizziert prototypisch realisierte Anwendungsszenarien mit CORBA-Anbindung der JavaCard und beschreibt die daraus gewonnenen Erfahrungen und das Entwicklungspotential für Java-fähige Chipkarten in netzbasierten Applikationen.

1 Einleitung

Als Ende der 60er- und Anfang der 70er-Jahre von Jürgen Dethloff und Roland Moreno Patente zur Einbringung einer Prozessorschaltung in eine Plastikkarte angemeldet wurden, ahnte noch niemand, wie vielseitig die Anwendungen von Chipkarten in Zukunft sein würden. In gut 20 Jahren entwickelte sich die Chipkarte, weitgehend unbemerkt von der Informatik-Fachwelt, vom reinen Datenspeicher (Speicherchipkarten) über einfache Spezialprozessoren (Mikrocontrollerkarten) hin zu vollwertigen, universellen Prozessoren – quasi Computer im Kleinstformat für die Brieftasche oder Geldbörse – mit derzeit ca. 32 kB Speicher, einer 32-Bit-RISC-Architektur und einigen MFLOPs Rechenleistung [Posegga1]. Damit entsprechen leistungsmäßig die Chipkarten den PC-Prozessoren vor einigen Jahren; ihnen fehlt zum vollwertigen Rechner eigentlich nur die Stromversorgung sowie die eigene Eingabe- und Ausgabemöglichkeit, beides wird daher gegenwärtig von den Kartenterminals bereitgestellt.

Chipkarten erlangten in letzter Zeit aufgrund ihrer einfachen, praktischen Handhabbarkeit sowie ihrer vielseitigen Verwendbarkeit eine starke Verbreitung; sie eroberten weite Bereiche des alltäglichen Lebens wie

[1] Diese Arbeit wurde von dem Zentrum für Kartenanwendungen und dem Technologiezentrum Darmstadt der Deutsche Telekom AG gefördert.

beispielsweise den elektronischen Zahlungsverkehr oder das Gesundheitswesen. Insbesondere eignen sie sich für den Einsatz als Sicherheitswerkzeug: Logische und physikalische Schutzmechanismen verhindern unautorisierte Zugriffe auf die in den Chips gespeicherten Daten oder geheimen Schlüssel, außerdem erlauben integrierte Krypto-Coprozessoren schnelle kryptographische Berechnungen. Chipkarten sind somit hervorragend als „portables“ Identifikations- und Authentisierungsmedium geeignet: Ihr Einsatz erstreckt sich vom persönlichen Profilträger und sicheren Behälter für persönliche Schlüssel und Zertifikate bis hin zur Ausführung von Chiffrier- und Signaturoperationen. Damit stellt die Smartcard eine wichtige technische Komponente für Anwendungen im Bereich digitale Signaturen, Electronic Commerce und Electronic Banking dar. Darüber hinaus zeichnet sich derzeit ab, daß Chipkarten als flexible System- und Ausführungskomponenten eine neue, zusätzliche Rolle in größeren, vernetzten Anwendungsszenarien zukommen wird.

Derzeit sind bereits mehrere hundert Millionen Chipkarten in Umlauf, wobei allgemein erwartet wird, daß der Trend auch in den kommenden Jahren unvermindert anhalten wird [Wohlmacher]: Kartenterminals als „Chipkartenannahmestellen“ sind mittlerweile als flächendeckende Infrastruktur vorhanden (Geldautomaten, öffentliche Telefone etc.), billige Lesegeräte für PCs beginnen den Massenmarkt zu erobern, erste Multifunktionsanwendungen werden im Kontext der Geldkarte erkennbar und Trendsetter wie beispielsweise Hersteller von Mobiltelefonen oder Anbieter von Mobilkommunikationsdiensten betreiben energisch die Weiterentwicklung der Chipkartenfunktionalität. Neue Entwicklungen wie die kontaktlose Chipkarte, die SuperSmartcard – eine Chipkarte mit integriertem Display und Keyboard – oder auch Hybridkarten, die verschiedene Kartentechnologien auf einer einzigen Karte vereinigen, bilden dabei die technischen Voraussetzungen für neue Einsatzmöglichkeiten und eine damit einhergehende weitere Verbreitung. Schätzungen zur Folge steigt die Zahl der produzierten Smartcards weltweit von heute ca. 750 Mio. Stück auf über 3 Mrd. bis zum Jahre 2003 [Kreft].

2 Software für Chipkarten

In gleichem Maße, wie die Verbreitung und Akzeptanz von Smartcards wächst, nehmen auch die Anforderungen zu. Komplexere Anwendungen, schneller aufeinanderfolgende Softwaregenerationen, aber auch der wachsende Konkurrenz- und Kostendruck bedingen neue Konzepte in der Chipkartenwelt. Forderungen nach nachladbarer Funktionalität und dynamisch erweiterbaren Applikationen, einfacherer und schnellerer Softwareentwicklung sowie Multiapplikationsfähigkeit ohne Kompromisse hinsichlich der Sicherheit gespeicherter Daten oder fremder Funktionseinheiten werden sowohl seitens der Chipkartenhersteller als auch der Kartenemittenten laut.

Entsprechend den technischen Gegebenheiten entwickelte sich in den letzten Jahren auch die Softwareseite weiter: Analog zur „Großrechnergeschichte“ vor ca. 25 Jahren findet gegenwärtig der Übergang von der Chipkartenrogrammierung in Maschinensprache zur Verwendung von Hochsprachen (wie beispielsweise Java) statt, mit z.T. um den Faktor 10 verkürzten Entwicklungs- und Projektdurchführungszeiten, sowie andererseits die Verwendung von (zwar abgespeckten, aber funktional weitgehend vollwertigen) Betriebssystemen (wie beispielsweise MULTOS und neuerdings auch „SmartCards for Windows“ von Microsoft), mit denen u.a multifunktionale Chipkarten ermöglicht werden, wodurch mehrere unterschiedliche Anwendungen unabhängig voneinander (bzw. in koordinierter Weise miteinander) auf einer Karte ablaufen können.

Durch die Verwendung von Hochsprachen und die Trennung von Anwendung und standardisierten Ablaufumgebungen in der Chipkarte eröffnet sich die Möglichkeit, die Applikationsentwicklung „outzusourcen“ und erzielt damit auch eine wesentlich verkürzte Projektdurchführungszeit: Anwendungsentwicklungen können nun vom Kartenemittenten bzw. Dienstanbieter selbst realisiert werden, wo bislang die Entwicklung von Hard- und Software meist noch in der Hand gefragter Chipkartenspezialisten mit spezifischen Systemkenntnissen lag.

2.1 MULTOS

MULTOS (Multiple Application Operating System; http://www.multos.com) bietet als erstes offenes und sicheres multiapplikationsfähiges Chipkarten-Betriebssystem die Möglichkeit, mehrere Applikationen nebeneinander auf einer Karte unterzubringen. MULTOS wurde ursprünglich von Mondex International unter Beteiligung von 17 Organisationen aus den verschiedensten Ländern entwickelt. Heute wird MULTOS von einem Konsortium verschiedener Kartenhersteller mit dem Namen MAOSCO weiterentwickelt. Erklärte Ziele von MAOSCO sind die Anerkennung von MULTOS als offener Industriestandard, die Weiterentwicklung und Pflege des Standards, das Management des Standardisierungsprozesses sowie die Bereitstellung eines Lizenzierungs- und Zertifikationsservices. MULTOS unterscheidet sich von der nachfolgend beschriebenen

JavaCard dadurch, daß es sich um ein Betriebsystem handelt, welches in integrierter Weise bereits die Multiapplikationsfähigkeit bereitstellt.

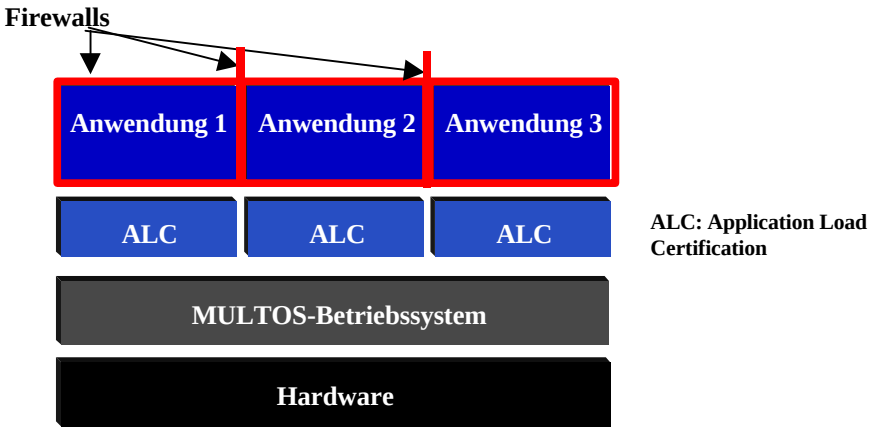

Abbildung 1: MULTOS-Architektur

Dem Anwendungsprogrammierer steht die Sprache „MULTOS Executable Language“ (MEL) und ein API, welches über eine C-Schnittstelle angesprochen werden kann, zur Verfügung. MEL ist eine Maschinensprache, die auf einer Application Abstract Machine (AAM) – der MEL Virtual Machine – abgearbeitet wird. Durch die Verwendung der AAM wird sichergestellt, daß Anwendungen eine definierte Schnittstelle zur Verfügung steht und sie somit unabhängig von der verwendeten Hardwareplattform sind. Diese Art der Plattformunabhängigkeit teilt die MULTOS Karte mit der auf Java basierenden JavaCard, die im folgenden vorgestellt wird.

2.2 Die JavaCard

Aufgrund ihrer flexiblen Nutzung und einfachen Einbindungsmöglichkeit in netzbasierte Anwendungsszenarien sind Chipkarten, die in Java, der objektorientierten „Internetsprache“, programmiert werden können besonders interessant. Eine JavaCard ist eine Smartcard, die neben einem beliebigen Betriebssystem einen Java Interpreter (eine sogenannte Java-Virtual-Machine (VM)) für den von Java-Compilern extern generierten Bytecode enthält. Die von der Java-VM zu unterstützenden Funktionen sind im „Java Card API“ [JavaCardAPI] definiert und legen i.w. die Kommunikation mit der Umgebung und den Zugriff auf Ressourcen der Karte (z.B. das Dateisystem) fest.

Java, eine Anfang der 90er Jahre von Sun Microsystems entwickelte Programmiersprache, der 1995 mit der stark gewachsenen Verbreitung des Internet der große Durchbruch gelang, entstand ursprünglich aus der Idee, eine Programmiersprache für Geräte aus dem Consumer-Bereich zu entwickeln, deren Code auf den unterschiedlichsten, in Anwendungssysteme eingebettete Prozessoren, von Set-Top-Boxen bis hin zur Kaffeemaschine, ausgeführt werden kann. Plattformunabhängigkeit und Portierbarkeit („Write Once, Run Everywhere“) wird insbesondere dadurch erreicht, daß Java nicht in Maschinencode übersetzt wird, sondern in Bytecode. Java unterstützt Multithreading und dynamisches Binden, d.h. Klassen werden erst dann geladen, wenn sie wirklich benötigt werden, wodurch es möglich ist, Klassenbibliotheken und damit Funktionalität zur Laufzeit zu erweitern, ohne daß laufende oder installierte Anwendungen im wesentlichen davon betroffen sind.

Die Verwendung von Java in einer Chipkarte ist in mehrererlei Hinsicht interessant, so beispielsweise hinsichtlich der einfacheren und schnelleren Anwendungsentwicklung auch durch Nicht-Chipkarten-Experten. Durch die Verwendung von Java wird außerdem die Interoperabilität mit externen, in Java realisierten Anwendungskomponenten unterstützt. Da Chipkarten vielfach im Kontext sicherheitsrelevanter Anwendungen eingesetzt werden (u.a. als vertrauenswürdige Ausführungsplattform), ist besonders dei in gewisser Weise bereits eingebaute Sicherheit und Robustheit von Java bemerkenswert, die durch die Verwendung eines die Ausführung „kontrollierenden“ Interpreters in Verbindung mit Bytecode-Verifikation [LY96] erreicht wird, auch wenn im Detail einige diesbezügliche Aspekte noch Forschungsgegenstand sind [Posegga2].

Obwohl Java für eingebettete Systeme entwickelt wurde, sind die Ressourcenanforderungen bezüglich Speicherkapazität und Prozessorleistung noch etwas zu hoch für heutige Chipkarten. Eine Einschränkung des Befehls- und Funktionsumfanges von Java war daher notwendig, um Java in Smartcards einsetzen zu können. Aus diesem Grund wurde im Februar 1997 von den führenden Kartenherstellern (Schlumberger, G&D, Gemplus, De La Rue, IBM, Bull, Keycorp, Oberthur, Motorola, ORGA, NEC, Toshiba) das JavaCard Forum (JCF) gegründet. Wichtigstes Ziel dieses Konsortiums ist die Definition eines für Chipkarten geeigneten Java-API, das sogenannte JavaCard API. Durch die Mitarbeit der Chipkartenbetriebssystemhersteller an der

Spezifikation soll die Interoperabilität gewährleistet werden. Weitere Ziele des JCF sind die Spezifikation der JavaCard VM einschließlich ihrer Eigenschaften, Austausch von Informationen bezüglich der Implementierung des JavaCard API zwischen den Mitgliedern, Erstellung technischer Dokumente, sowie die Unterstützung von Software- und Hardwareentwickler im Kontext der JavaCard.

Der aktuelle JavaCard 2.0-Standard definiert die Grundfunktionalität einer Java-Chipkarte, das sogenannte JavaCard Runtime Environment. Dieses besteht aus der JavaCard Virtual Machine (VM) sowie dem JavaCard API (siehe Abb. 2). Der Sprachumfang von Java wurde dabei auf die für Chipkartenanwendungen notwendigen Funktionen reduziert. JavaCard 2.0 unterstützt daher kein dynamisches Nachladen von Klassen, kein Multithreading, keine mehrdimensionalen Felder und auch nicht das Clonen von Objekten. Von den Datentypen werden float, double und long nicht unterstützt. Aus Kapazitätsgründen wird bei gegenwärtigen Implementierungen oft auch kein Garbage Collection durchgeführt.

Zwar ist nun aufgrund der knappen Ressourcen gegenwärtiger Chipkarten der Sprachumfang etwas eingeschränkt, jedoch tut das der Verwendung von Java für typische Applikationen i.a. keinen Abbruch: Abgesehen von diesen Einschränkungen laufen Java-Programme (bzw. „Applets“ oder im Kontext von Chipkarten auch „Cardlets“ genannt) überall dort, wo eine Java-VM vorhanden ist: In einem Server, einem PC, einem WWW-Browser oder eben einer Smartcard. Im übrigen ist davon auszugehen, daß sich sowohl der nutzbare Speicher als auch die Prozessorleistung von Smartcards in den nächsten Jahren noch wesentlich steigern läßt, so daß die heute noch existierenden Einschränkungen im Sprachumfang aufgehoben werden können. Ferner wird erwartet, daß mit der Massenproduktion die Kosten für typische JavaCards von derzeit ca. 20 $ auf ca. 5 $ fallen werden. In diesem Zusammenhang ist auch interessant, daß Sun schon vor einiger Zeit einen Chip angekündigt hat, der Java-Bytecode direkt ausführen kann.

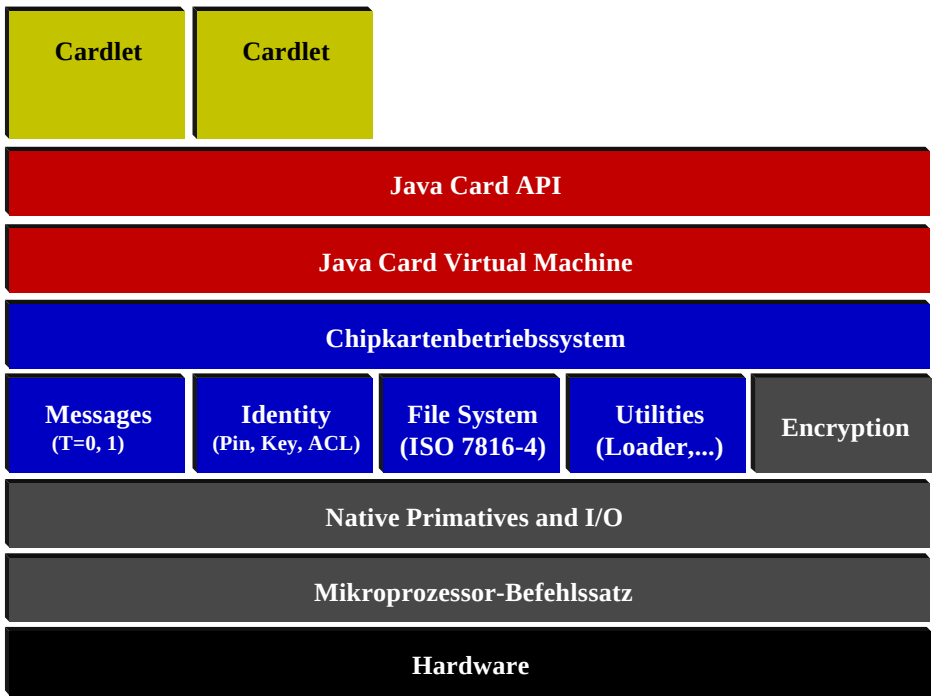

Abbildung 2: JavaCard-Softwarearchitektur

3 Die JavaCard als Infrastrukturkomponente in verteilten Anwendungen

Verteilte Applikationen haben heute, wo oft das Internet die zugrundeliegende „Plattform“ darstellt, eine ganz andere Dimension als noch vor kurzem. Dies macht die Verwendung einer neuen Art von Middleware als Infrastruktur für solche Applikationen erforderlich, in die auch die Chipkarte als eine Komponente, die innerhalb eines solchen Systemverbundes eine Teilfunktionalität erfüllt, eingebettet werden sollte. Gegenwärtig ist allerdings zu beobachten, daß das Internet nicht nur weiter stetig wächst, sondern auch dynamischer und mobiler wird, was nicht ohne Auswirkungen für Infrastrukturaspekte bleiben wird. Dies hat mehrere Gründe:

- Das Internet selbst wächst nicht nur hinsichtlich der Zahl der Nutzer und angeschlossenen Rechner, sondern verästelt sich beispielsweise bis hin zu einfachen Haushaltsgeräten. Hierzu tragen auch über das Stromnetz vermittelte Kommunikationsmöglichkeiten und auf preiswerten Chips realisierte „eingebettete“ Java-VMs bei, die in Kürze vermehrt zu erwarten sind.

- An das Internet angeschlossene Geräte werden (im Vergleich zu traditionellen Rechnern) zunehmend „exotischer“ und weisen ein ungewöhnliches Verhalten auf. So sind PDAs und Mobiltelefone nur zeitweise erreichbar und wechseln mit dem physischen Ort auch ihre Umgebung und damit die lokal vorhandenen Dienste. Auch Chipkarten stellen solche eher unkonventionellen Geräte dar, die u.U. sogar nur kurzlebige Verbrauchsgüter verkörpern.

- Dienste emanzipieren sich immer mehr von den eigentlichen, sie klassischerweise tragenden Geräten („Server“) und werden dadurch virtueller: diese haben ggf. nicht nur keinen konkreten Ort mehr, sondern auch keine verantwortliche Instanz – z.B. dann, wenn Dienstleistung in einem wechselseitigen Verbund erbracht wird und dadurch einen Mehrwert realisiert. Die traditionelle Client/Server-Struktur löst sich damit teilweise auf.

- Dienste und Dienstleistungen gewinnen in ökonomischer Sicht eine immer größere Bedeutung, wohingegen Internet-fähige Geräte immer billiger werden. Es ist vorstellbar, daß schon bald solche Geräte in großem Umfang gratis abgegeben werden, um Umsatz mit Diensten zu generieren.

Die oben genannten Aspekte, zusammen mit der technisch bedingten Weiterentwicklung (drahtlose Netze wie Bluetooth [Bluetooth], höhere Kommunikationsleistung, weitere Miniaturisierung etc.) und ökonomisch-politischen Gegebenheiten (Offenheit der Schnittstellen, Interoperabilität, kürzere Produktzyklen, Globalisierung der Märkte etc.) haben zur Konsequenz, daß Applikationen in einem verteilten Systemverbund offener, dynamischer und prinzipiell umfänglicher werden als traditionelle netzbasierte Anwendungen.

Als mittlerweile „klassische“ Middleware-Plattform stellt CORBA einen Infrastrukturrahmen zur Unterstützung von Applikationen bereit, welche aus ggf. weiträumig verteilten miteinander kooperierenden Objekten bestehen. Herkömmliche verteilte Architekturprinzipien sind für große, sehr dynamische und völlig offene Systeme jedoch zu wenig adaptiv und daher eher ungeeignet. Mit Java wurde eine Sprache konzipiert, die von ihren Konzepten her besser für das Internet und seine neuen Anwendungen geeignet ist als klassische Programmiersprachen. Allerdings gehört zu einer guten „Internet-Infrastruktur“ wesentlich mehr als eine Sprache: Gefordert sind entsprechend geeignete Ablaufumgebungen, Betriebssysteme und Verteilungsplattformen. Bezüglich Java hat Sun kürzlich mit Jini [Jini] ein auch für diesen Zweck hochinteressantes Framework vorgestellt. Allerdings steht Sun hier nicht alleine, so verfolgt etwa IBM mit T-Spaces ein ähnliches Konzept [T-Spaces].

Es sieht daher so aus, als ob man am Anfang einer neuen Generation von „Internet-Middleware“ steht, die durch ihre Mechanismen und Paradigmen die Dynamik und Offenheit der sich abzeichnenden großen verteilten Applikationen besser unterstützt. Andererseits steht ein fast alles durchdringendes Internet mit seinen vielfältigen interoperablen Diensten und die propagierte Offenheit eines solchen Systems, das teilweise auch Verantwortlichkeiten virtualisiert, zunächst im Gegensatz zu Sicherheit, Privacy und Vertrauenswürdigkeit. Nun setzen sich gerade Chipkarten in großem Stile als Instrumente zur Erhöhung von Sicherheitsbelangen durch. Interessant ist daher, welche Rolle Chipkarten in der sich abzeichnenden wesentlich offeneren und dynamischeren Welt, für die die neue Internet-Middleware konzipiert ist, spielen können. Gerade die JavaCard ist in dieser Hinsicht bemerkenswert, da sie Programme in einer relativ sicheren (dem Benutzer gehörenden und in veritabler Hinsicht „portablen“) Umgebung ausführen kann. Für mobilen Code und Szenarien aus dem Bereich des Electronic Commerce scheint die JavaCard daher prädestiniert zu sein; durch ihre eingebaute virtuelle Java-Maschine und die Möglichkeit, dynamisch Programmkomponenten aus dem Netz zu laden, stellt sie zudem (relativ zur „Java-Welt“) eine universelle Plattform in einer offenen, verteilten Welt dar.

Jedenfalls wird deutlich, daß, um Chipkarten in einem offenen Verbund mit anderen Geräten und Services betreiben zu können, in jedem Fall eine Eingliederung in eine standardisierte Middleware-Architektur notwendig wird. Im nachfolgenden Kapitel wird daher u.a. skizziert, wie eine Anbindung der JavaCard an die CORBA-Infrastruktur realisiert werden kann. Die weitergehende Verwendung der JavaCard als sichere Ausführungsumgebung für mobilen Code ist Gegenstand eines anderen Projektes [KIVS99].

3.1 Erfahrungen bei der prototypischen Realisierung einiger JavaCard-Applikationen

An der TU Darmstadt wurden am Fachbereich Informatik vom Fachgebiet „Verteilte Systeme“ und dem Informationstechnologie-Transfer-Office (ITO) in Zusammenarbeit mit J. Posegga und H. Vogt vom Technologiezentrum der Deutschen Telekom AG in Darmstadt einige Applikationsszenarien prototypisch realisiert, um das Potential der JavaCard beim Einsatz in zukünftigen Diensten und verteilten Anwendungen aus dem Bereich der Telekommunikation zu untersuchen. Beispielhaft dafür wird nachfolgend die CORBA-Anbindung sowie die Realisierung einer Lotto-Kundenkarte beschrieben.

3.1.1 Middleware-Integration: CORB

Middlewareplattformen finden als Basis für verteilte Applikationen immer weitere Verbreitung. Die Vorreiterrolle wird hier von der OMG-Organisation übernommen, die als Standardisierungsgremium die Spezifikationen für CORBA festlegt. Kernstück jeder CORBA-konformen Middleware ist der sogenannte *Object Request Broker* (ORB), der die Vermittlung zwischen räumlich getrennten Objekten übernimmt. Dabei bietet CORBA einem Anwendungsentwickler den Vorteil, daß bei der Anforderung einer Objektreferenz der ORB ein passendes Objekt sucht, gleichgültig wo es sich befindet oder in welcher Programmiersprache es implementiert ist.

Eine Middelwareplattform wie CORBA enthebt den Programmierer netzbasierter Anwendungen von vielen Details, die durch die verteilte Natur der Applikation entstehen. Dies kann man sich auch im vorliegenden Kontext zunutze machen: Ein Applet, das auf der JavaCard abläuft, benötigt typischerweise einen lokalen Kommunikationspartner auf dem Rechner, an den der Kartenleser angeschlossen ist, der die in der Regel passive Karte anspricht. Einmal angestoßen, kann die Applikation in Form des auf die Karte geladenen Applets es erfordern, daß sich das Applet mit einem im Netzwerk angesiedelten Dienst verbindet. Zur Zeit muß dazu noch eine Vermittlungsinstanz im Kartenterminal benutzt werden, die dann die eigentliche Verbindung zum Netzdienst aufbaut. Dieser Netzdienst ist idealerweise CORBA-basiert, so daß ein im Netz befindlicher ORB von der Vermittlungssoftware angesprochen werden kann.

Um den Aufwand, der durch die jeweils applikationsspezifisch zu realisierende Vermittlungsinstanz entsteht zu eliminieren, sollte die ORB-Funktionalität einem Karten-Applet direkt auf der Karte zur Verfügung stehen. Aufgrund der zur Zeit noch beschränkten Ressourcen auf den Karten kann die Integration und Bereitstellung eines solchen Card-ORB (CORB) in zwei Phasen vollzogen werden:

1. Der Teil des CORB, die die Vermittlungsfunktionalität realisiert, ist auf den Rechner, an den die Karte angeschlossen ist, ausgelagert. Das Kartenterminal dient damit als zusätzliche Vermittlungsinstanz (siehe Abbildung 3). Die Klassen, die den CORB auf der Karte selbst realisieren, sind dabei Proxy-Objekte, die die Funktionaliät des extern angesiedelten CORB-Teiles in die Karte spiegeln, in dem die Objektreferenzen und Dienstanfragen dorthin weitergereicht werden. Dabei muß für die eigentlichen Dienstobjekte in gleicher Weise verfahren werden: Die Funktionalität der Dienst-Stellvertreter (Proxys) wird funktional aufgespalten in einen Teil, der auf dem Kartenterminal angesiedelt ist (Diens-Proxy), und einen Teil, der in der Karte selbst abläuft (Dienst-CardProxy).

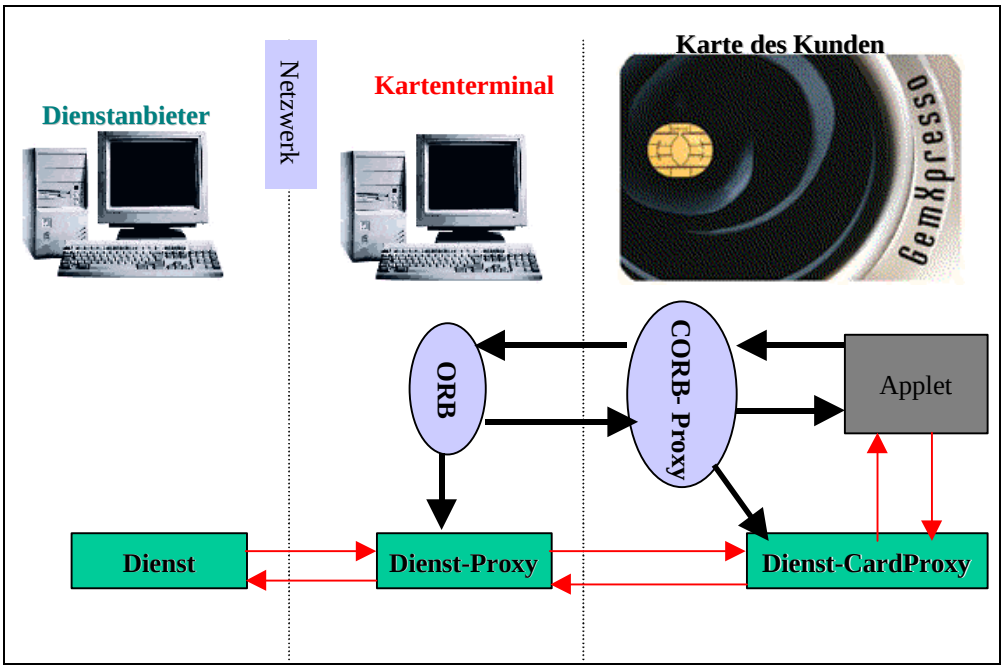

Abbildung 3: Einfaches ORB-Szenario

2. Die Vermittlungsfunktionalität ist vollständig durch die CORB-Klassen auf der Karte implementiert (siehe Abbildung 4).

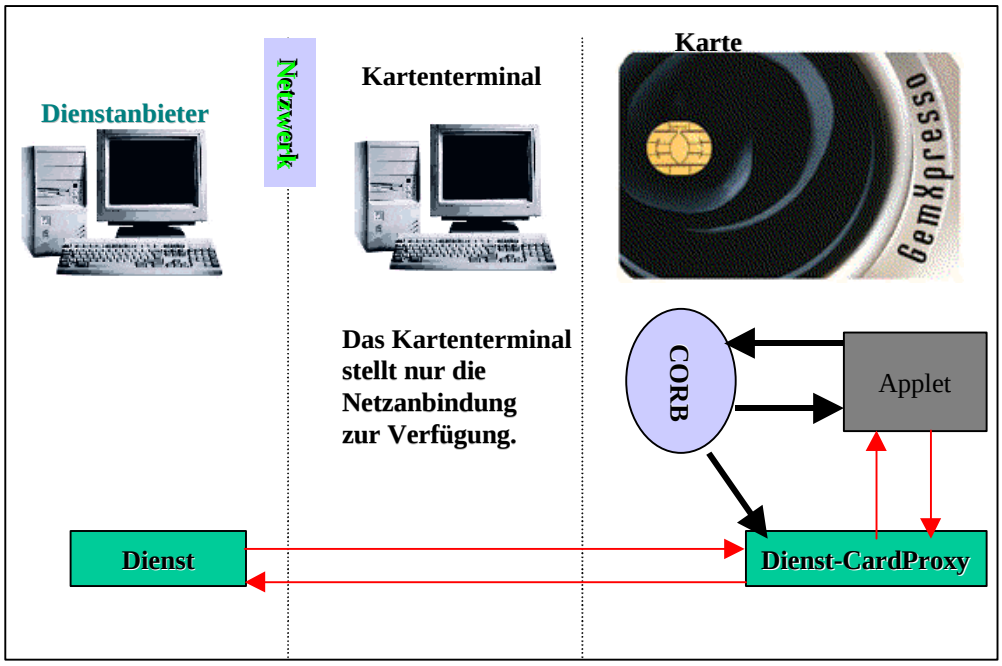

Abbildung 4: Ideales ORB-Szenario

Läuft die gesamte ORB-Funktionalität auf der Karte ab, so kann man den Kartenapplikationen einen vertrauenswürdigen ORB anbieten, da die Karte als sichere Ablaufumgebung angesehen werden kann. Dieser ORB könnte zusätzlich vom Kunden nach seinen Präferenzen personalisiert werden, was die Sicherheit beim Arbeiten mit unbekannten externen Komponenten erheblich erhöht: So ist denkbar, daß vom Benutzer die ansprechbaren Netzobjekttypen eingeschränkt werden, oder seine Konfiguration besagt, daß nur verschlüsselt mit Objekten kommuniziert werden darf.

Die Implementation der gesamten ORB-Funktionalität auf der JavaCard entsprechend Abbildung 4 ist zur Zeit aus Platz- und Performanzgründen so jedoch noch nicht realisierbar. In diese Phase kann erst mit der Verfügbarkeit leistungsfähigerer Karten eingetreten werden. Die Realisierung einer einfachen CORB-

Implementation durch Objekte, die entsprechend Abbildung 3 die externe ORB-Funktionalität lediglich in die Karte spiegeln, ist hingegen unserer Erfahrung nach mit den zur Zeit erhältlichen Karten möglich. Neben den Klassen, die die ORB-Funktionalität anbieten, sind zusätzlich Klassen nötig, die als Proxy für die Objekt-Stubs fungieren (Dienst-CardProxy Objekte in Abbildung 3).

Bei der Nutzung der neuen Infrastruktur, etwa in einem Homebankingszenario, in dem die Homebankingfunktionalität als Applet auf der Karte abläuft, wird dann als Kommunikationspartner ein Banking-Serverobjekt der Bank angesprochen. Dieses Objekt kann nun der Applikation auf der Karte per CORBA verfügbar gemacht werden. Das Applet muß lediglich die Referenz vom lokalen CORB anfordern, um dann über diese Referenz mit dem entfernten Serverobjekt so zu kommunizieren, als ob es lokal verfügbar wäre.

Die Vorteile dieser Architekur liegen klar auf der Hand: Karten-Applets können in einfacher und flexibler Weise auf *beliebige* Netzdienste zugreifen, indem sie alle Möglichkeiten einer CORBA-basierten Middleware-Plattform nutzen. Dadurch können einem Kunden als Nutzer der JavaCard beliebige Dienstleistungen über seine Karte angeboten werden; dabei ist es gleichgültig, wo sich der Kunde gerade befindet, bzw. wo der Netzdienst angesiedelt ist. Dadurch, daß (idealerweise) die komplette ORB-Funktionalität auf der Karte abläuft, kann der Kunde dieser Vermittlungsinstanz Vertrauen entgegenbringen. Durch die Möglichkeit, den CORB zu personalisieren, schafft man zusätzliche Schutz- und Sicherungsmaßnahmen. Für den Kartenherausgeber ergibt sich ein Zusatznutzen dadurch, daß mit dem Vorhandensein einer CORBA-Plattform auf der Chipkarte selbst eine weite Palette von Dienstleistungsangeboten verfügbar wird. Neue Dienste können ohne viel Aufwand in Kartenapplikationen integriert und selbst von der Chipkarte aus angesprochen werden.

3.1.2 **LottoCard – eine ubiquitäre Kundenkarte**

Die LottoCard ist ein Beispiel für eine maßgeschneiderte Privatkundenkarte, die sich die vorgestellte CORB-Infrastruktur zunutze machen könnte[2]. Realisiert wurde die LottoCard durch eine entsprechend programmierte JavaCard. Das Szenario stellt sich wie folgt dar:

Der heutige Lotto-Kunde geht zu einer der Lotto-Annahmestellen, füllt umständlich seinen Tipschein Woche für Woche erneut aus, dieser wird maschinell eingelesen und digital an die Lottozentrale geschickt. Der Kunde überprüft seinen Lottoschein selbst manuell auf Gewinn oder maschinell bei der Annahmestelle. Um den Gewinn abzuholen, muß er jedoch auf jeden Fall persönlich zur selben Lotto-Annahmestelle gehen, bei der er auch den Tip abgegeben hat. Diese Prozedur läßt sich durch den Einsatz einer Smartcard einfacher und kundenfreundlicher gestalten.

In unserem prototypisch realisierten Szenario erhält der Lotto-Kunde seine persönliche LottoCard. Mit dieser kann er nun überall auf der Welt zu jeder beliebigen Zeit Lotto spielen. Einzige Voraussetzung ist ein Telefonanschluß (Handy, Telefonzelle, einfaches Telefon oder PC mit Modemanschluß) mit integriertem Kartenlesegerät. Erwirbt der Kunde seine Kundenkarte, so wird für ihn ein anonymes Lottokonto mit Kontonummer und Paßwort angelegt (Personalisierung). Über seine Nummer, die auf der PIN-geschützten LottoCard gespeichert wird, und seinem Paßwort kann der Kunde Geld zum Lottospielen laden, wie auch gutgeschriebene Gewinne abbuchen. Die Umbuchung kann sowohl von einem Girokonto, einer Kreditkarte oder einer elektronischen Geldbörse (GeldKarte, PayCard), die als zusätzliche Applikation auf seiner LottoCard zur Verfügung steht, erfolgen.

[2] Das LottoCard Szenario wurde ohne Nutzung der CORB-Infrastruktur entwickelt. Eine einfache Variante des Szenarios wurde später auf der CORB-Infrastruktur implementiert.

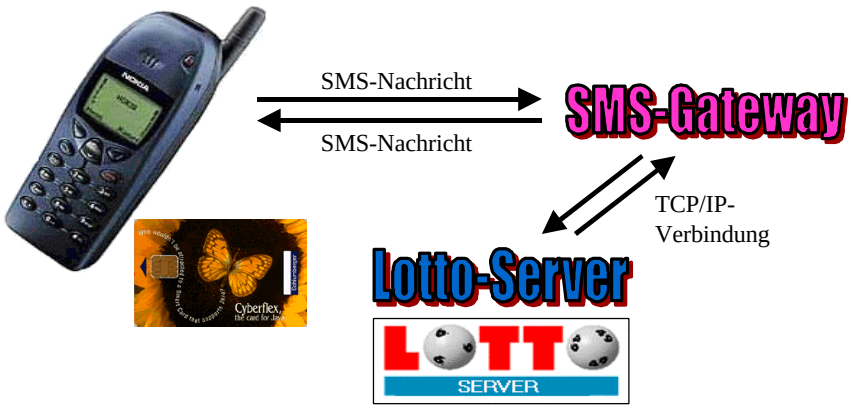

Abbildung 5: Architektur des LottoCard-Szenarios

Um die Sicherheit bei der Übertragung von Daten von der Karte zum Lottoserver über öffentliche Netze (z.B. Internet) zu gewährleisten, enthält die LottoCard kryptographische Schlüssel zur Authentisierung und Verschlüsselung der Kommunikation. Diese Schlüssel werden bei der ersten Kommunikation der Chipkarte mit dem Lottoserver über das Diffie-Hellman-Protokoll vereinbart. Über ein Challenge-Response-Verfahren und den geheimen Schlüssel wird zu Beginn jeder Kommunikation die Identität beider Kommunikationspartner überprüft. Jede weitere Kommunikation zwischen Karte und Server wird in der aktuellen Implementierung mittels symmetrischer Verschlüsselung gesichert.

Um Lotto spielen zu können, muß der Kunde nun seine LottoCard in das Kartenlesegerät seines Telefons stecken und die Lotto-Applikation anwählen. Die Lotto-Applikation wählt nun den Lottoserver eigenständig an und führt eine gegenseitige Authentisierung mit dem Server durch. Nun kann der Kunde am Telefondisplay über Tasteneingabe wählen zwischen Kontoinformationen abfragen, Geld auf- und abbuchen, Lottotip abgeben oder letzten Tip auf Gewinn überprüfen und anzeigen lassen.

Das Eingeben eines Lottotips erfolgt über die Telefon- oder PC-Tastatur und kann über bereitgestellte Menüs an den Lottoserver gesandt werden. Dieser nimmt den signierten, digitalen Lottoschein entgegen und sendet an die LottoCard eine entsprechende seinerseits signierte Quittung, die auf der LottoCard gespeichert wird. Somit hat der Lotto-Kunde einen rechtlichen Nachweis, daß er einen bestimmten Lottoschein abgegeben hat. Bei der Gewinnüberprüfung wird die Kennung des Kunden an den Server geschickt. Dieser kann über die Kundenkennung den gespeicherten Lottoschein ermitteln und auf Gewinn überprüfen. Der Lottoserver sendet nun die Gewinndaten zusammen mit den gezogenen Lottozahlen an die LottoCard, die diese Daten dem Lotto-Kunden am Display anzeigt. Da sämtliche Lottodaten auf der LottoCard gespeichert sind, kann der Lottospieler beliebige, ältere Tips erneut abgeben, statistische Auswertungen über gezogene Zahlen oder seine Tips vornehmen oder für eine Langzeitspeicherungen von der Karte in seinen PC laden.

Eine Sonderstellung nimmt das Handy als Endgerät ein, da dieses als einzige Möglichkeit zur Kommunikation mit einem Server im Telekommunikationsnetz SMS-Messages zur Verfügung hat. Aus diesem Grund ist in diesem Szenario gegenüber den Telefon-Szenarien ein zusätzlicher SMS-Server im Netz notwendig, der die SMS-Messages vom Handy entgegennimmt und über TCP/IP-Kommunikationsprotokolle an den Lottoserver weiterleitet, Nachrichten vom Lottoserver annimmt und diese in SMS-Nachrichten umgewandelt an die LottoCard im Handy sendet (siehe Abb. 3).

3.2 Erforderliche Infrastrukturmaßnahmen

Damit Applikationen von einer verteilten Anwendungsarchitektur, in die die Java-Chipkarte eingebettet ist, profitieren können, müssen eine Reihe von infrastrukturellen Voraussetzungen erfüllt sein, die die Nutzung nicht nur vereinfachen, sondern im wesentlichen erst ermöglichen. Dabei lassen sich die nötigen Infrastrukturmaßnahmen in zwei Kategorien unterteilen:

1. Infrastrukturmaßnahmen auf den Karten selbst, und
2. Infrastrukturmaßnahmen zur Anbindung der Karten an bestehende Architekturen.

Die Maßnahmen der beiden Kategorien können dabei in der Regel nicht isoliert umgesetzt werden.

In einem idealen Szenario sollte eine Smartcard nicht anders behandelt werden als ein beliebiger Netzknoten, allerdings mit dem Unterschied, daß der „Smartcard-Rechner“ den Applikationen bzw. den

Applikationsentwicklern spezielle Eigenschaften bereitstellen kann. Soll eine Smartcard wie ein beliebiger Rechner in einem Intranet oder im Internet angesprochen werden können, so ist die Möglichkeit der direkten Kommunikation mit der Karte essentiell. Da Kommunikation im allgemeinen und insbesondere Kommunikationsplattformen, sogenannte Middleware, grundlegende Mechanismen zur Realisierung verteilter Anwendungen bilden, liegt es nahe, die Integration von Chipkarten in diese Architekturen zu betrachten, um die Smartcards so in uniformer Weise zugreifbar und nutzbar zu machen.

Es stellt sich heraus, daß gegenwärtige Betriebssysteme für Karten, die Middlewareunterstützung, aber auch die grundlegenden Architekturprinzipien (z.B. Master-Slave-Rolle) einige Defizite aufweisen. Im folgenden werden die diesbezüglich gewonnen Erfahrungen aus den von uns realisierten Beispielszenarien dargestellt und die nötigen Infrastrukturmaßnahmen vorgestellt und diskutiert.

3.2.1 **Kartenlokale Infrastruktur**

Idealerweise wird die Smartcard als ein eigenständiger Rechner im Netzverbund angesehen. Damit eine Karte so behandelt werden kann, muß sie jedoch eine ähnliche lokale Infrastruktur aufweisen wie ein herkömmlicher Rechner. Neben der rein physikalischen Ausstattung mit CPU und Speicher muß insbesondere das Betriebssystem auf die im Vergleich zur bisherigen Nutzung neuen Aufgaben eingerichtet sein. Hier bieten auch die vorgestellten neueren Kartenbetriebsysteme kaum adäquate Unterstützung: Es ist zwar möglich, neuen ausführbaren Code fast jederzeit auf die Karte zu laden, bisher kann aber lediglich genau eine der auf der Karte befindlichen Applikationen aktiv sein. Zwar gibt es prinzipiell die Möglichkeit, beim Start der Karte die Aktivierung einer bestimmten Applikation zu fordern, dieses Protokoll wird jedoch in der Praxis bisher kaum von Kartenterminals, also den Systemen, an die Kartenleser und Karten angeschlossen sind, benutzt.

Neben diesen rein organisatorischen Mängeln hat sich im Rahmen der prototypischen Realisierungen der obigen Beispielszenarien gezeigt, daß die zur Zeit erhältlichen Betriebsysteme einen gravierenden Nachteil besitzen: es ist bisher nicht vorgesehen, daß die Karte von sich aus aktiv wird. Den Betriebsystemen fehlt dazu die bei Rechnern übliche interruptgesteuerte Verarbeitung. Dies liegt darin begründet, daß bisher Smartcards lediglich als Server zur Bereitstellung einer bestimmten Funktionalität angesehen wurden. Dabei geschieht die Bereitstellung dergestalt, daß der auf der Karte befindliche Server-Applikationscode bei jeder Anfrage von Anfang an neu ausgeführt wird. Mit der auf die Anforderung folgenden Antwort von der Karte gilt der Dienst als erbracht. Ein weiteres Problem stellt die beschränkte Kommunikationsbandbreite zwischen Karte und der Applikation, die die Karte nutzt, dar. So können die zur Antwort nötigen Daten u.U. nicht in einer einzigen Antwortnachricht an die Applikation übertragen werden. Aus diesem Grund sieht das bisher vorherrschende Kommunikationsprotokoll zwischen Karte und Applikation vor, das Ergebnis in mehreren Schritten paketisiert zu versenden. Dazu wird in der ersten Antwortnachricht der Karte eine Markierung gesetzt, daß noch weitere Daten von der Karte anzufordern sind, um die Gesamtantwort zu erhalten. Die die Karte nutzende Applikation kann die wartenden Datenpakete dann jeweils mit einer speziellen Anforderung von der Karte beziehen.

Sollen auf der Karte nun Client-Server-basierte Applikationen ablaufen, die von sich aus Anfragen (etwa Funktions- und Methodenaufrufe oder gar Netzverbindungen) an die die Karte nutzende Applikation oder entfernte Netzobjekte stellen, so kann der obige Mechanismus zwar benutzt werden, um ein solches Verhalten zu simulieren, verlangt aber eine spezielle und aufwendige Programmierung der Kartenapplikation, und ist damit nicht transparent für den Applikationsentwickler. Dies zeigt, daß neue Mechanismen, wie z.B. Interrupts, in den Kartenbetriebsystemen nötig sind.

Bei der Verwendung von Smartcards als Anwendungsplattform sollte es auch möglich sein, nicht nur eine einzige Applikation zu einem Zeitpunkt auf der Karte auszuführen, sondern mehrere Anwendungen nebenläufig zueinander. Dies erfordert allerdings weitere Vorkehrungen im Kartenbetriebssystem. Laufen Applikationen parallel, so muß die Ressource CPU-Zeit unter den Applikationen aufgeteilt werden, was einen Scheduler erfordert. Daneben wird dann aber auch die Verwaltung des Speichers notwendig, wie man sie in jedem herkömmlichen Rechner vorfindet. Die den einzelnen Applikationen zugeordneten Speicherbereiche müssen gegeneinander abgeschottet werden, um illegale Speichermanipulationen zu verhindern. Dabei muß sich der Speicherschutz auch auf die Bereiche beziehen, die das Dateisystem der Karte bilden. Diese sollten mit der Möglichkeit zum Setzen von geeigneten Zugriffsberechtigungen ausgestattet sein.

Da Smartcards aufgrund ihrer Beschaffenheit vielfach in sicherheitskritischen Bereichen und Applikationen eingesetzt werden, sind die meisten Smartcards mit Coprozessoren ausgestattet, die die in diesem Bereich anfallenden kryptographischen Berechnungen effizient ausführen. Im Rahmen einer Benutzung als Anwendungsplattform muß die Funktionalität des Krypto-Coprozessors dem Programmierer in geeigneter Form,

zum Beispiel als Programmbibliothek, angeboten werden. Aufgrund der verschiedenen Exportvorschriften in den Ländern der Kartenhersteller wird diese Unterstützung von verschiedenen Karten zur Zeit nicht oder nur in eingeschränkter Form angeboten. In zukünftigen Szenarien könnten die Karten netzweit ihre spezifischen Fähigkeiten exportieren und zur Verfügung stellen. Dies macht allerdings standardisierte Schnittstellen und Zugriffsmechanismen nötig, wie sie zum Beispiel in Jini angedacht sind.

Zusammenfassend läßt sich sagen, daß die aktuellen Chipkarten mit ihren Betriebssystemen noch nicht die Fähigkeit einer universellen Nutzung in einem Netz anbieten. Damit die Vision einer effizienten und netzglobalen Nutzung Wirklichkeit werden kann, müssen insbesondere die Kartenbetriebssysteme eine ähnliche Entwicklung durchlaufen wie die Betriebssysteme herkömmlicher Rechner. Neben performanter Kartenhardware wird zumindest Speichermanagement, Speicherschutz und Scheduling benötigt. Es ist zu erwarten, daß diese Mechanismen im Zuge der weiteren Evolution in Zukunft auf vielen Smartcards zu finden sein werden.

3.2.2 Infrastruktur außerhalb der Karte

Um eine Verfügbarkeit von Smartcards als allgemeine Programmierplattform in einem offenen Netzverbund zu erreichen, sind nicht nur die oben beschriebenen Anforderungen an die Karten selbst zu erfüllen, sondern auch Infrastrukturmaßnahmen im Netz bzw. auf dem Rechner, an den die Karte über einen Kartenleser angeschlossen ist, durchzuführen.

Auf die Chipkarte kann nur mittels Kartenleser zugegriffen werden (wobei das Lesegerät bei speziell dafür vorgesehenen Karten auch kontaktlos mit der Karte kommunizieren kann). Das Kartenlesegerät ist dabei in aller Regel selbst an einen Rechner (Kartenterminal) angeschlossen. Damit muß jegliche Kommunikation zwischen Karte und einer nicht lokalen Applikation durch den Kartenrechner vermittelt werden. Dies macht aber zusätzliche Vermittlungssoftware auf dem Kartenrechner nötig. Damit der Zugriff und die Kommunikation mit verschiedenen Kartentypen und Kartenlesertypen in uniformer – und im Idealfall in transparenter – Weise geschehen kann, muß hier eine entsprechende Schnittstelle zur Verfügung gestellt werden, die einerseits die Ressourcen der Karte, und andererseits die Applikationen auf der Karte netzweit anbieten kann. Diese Anbindung kann, wie bisher meist üblich, applikationsspezifisch geschehen – hier existiert zu jeder Kartenapplikation ein entsprechender Kommunikationspartner auf dem Kartenrechner – sollte allerdings zweckmäßigerweise applikationsunabhängig sein und generischen Charakter besitzen.

Da in Form von Middleware bereits ähnliche, generische Schnittstellen und Mechanismen für klassische Rechnerverbundsysteme existieren (CORBA, JINI, DCOM etc.), liegt es nahe, die existierenden Mechanismen zu nutzen und die Kartenanbindung dort zu verankern. Wie die oben beschriebene prototypische Anbindung der Karte an die CORBA-Middleware zeigt, ist dies zur Zeit nur mit Einschränkungen möglich: hier muß noch eine Vermittlungsinstanz auf dem Kartenrechner eingesetzt werden. Es zeichnet sich allerdings ab, daß die Leistungsfähigkeit der auf Chipkarten verfügbaren Ressourcen ähnlich schnell zunehmen wird, wie dies im Bereich herkömmlicher Rechnertechnologie geschieht. Zusammen mit den zu erwartenden Weiterentwicklungen der Kartenbetriebssysteme kann die Anbindung dann auch direkt auf der Karte geschehen, so daß der Kartenrechner lediglich noch den physikalischen Netzanschluß für die Karte bereitstellt. Im Extremfall kann der Netzanschluß auch nur durch den Kartenleser geschehen, so daß die Karte direkt über das Netz angesprochen werden kann. Aufgrund der zu erwartenden massenhaften Verbreitung und damit sehr großen Zahl von Chipkarten sind die heutigen Verfahren der Netzkomponentenadressierung dafür allerdings nicht mehr geeignet, da sie keinen ausreichend großen Adreßraum anbieten. Abhilfe kann hier der kommende Internetstandard IPv6 bieten, der durch die Verwendung neuer Adreßierungsverfahren, und damit einer fast unbegrenzten Anzahl verschiedener Netzadressen die Möglichkeit eröffnet, jede Chipkarte mit einer eigenen IP-Adresse zu versehen.

3.3 Interoperabilität

JavaCards sind von verschiedenen Herstellern erhältlich. Der einfache Einsatz der Karten im Rahmen verteilter Anwendungsszenarios wird wesentlich von der Möglichkeit bestimmt, JavaCard-Applikationen auf jede beliebige JavaCard zu laden und dort auszuführen, gleichgültig von welchem Hersteller die Karte stammt. Dies ist derzeit jedoch problematisch. Zwar arbeiten die im JavaCard-Forum zusammengeschlossenen Hersteller an einem gemeinsamen Standard, der die Interoperabilität sichern soll, bisher ist diese aber noch nicht ganz erreicht.

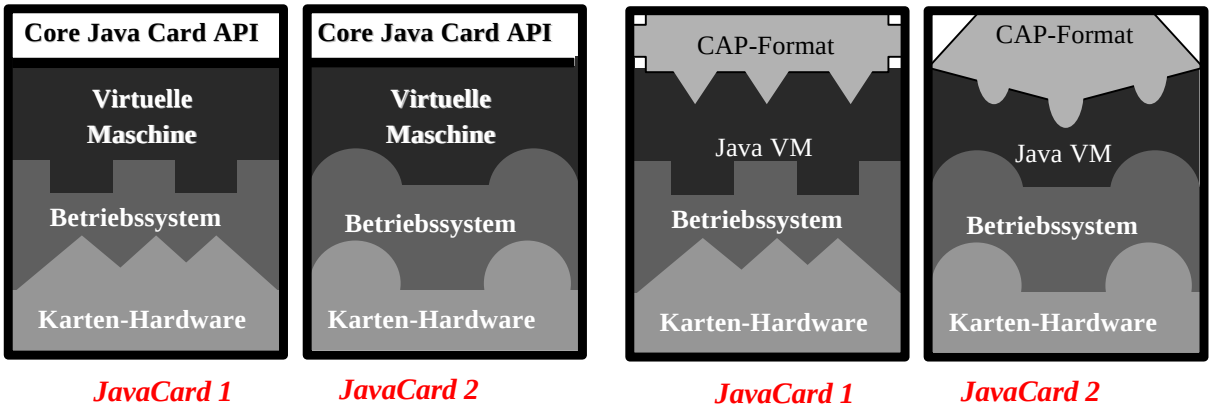

Abbildung 6: Karteninteroperabilität

Die JavaCards der verschiedenen Hersteller bauen auf verschiedenen Hardwareplattformen auf und benutzen dementsprechend auch unterschiedliche Kartenbetriebssysteme (siehe Abbildung 6). Auf dem Betriebssystem setzt die jeweilige Implementation der Java VM auf. Diese bietet über das JavaCard API eine standardisierte und damit einheitliche Programmierschnittstelle an, die von den Kartenapplikationen genutzt werden kann. Der ausführbare Code der Applikation soll dann vom Kartenprozessor ausgeführt werden. Allerdings ist dieser plattformunabhängige sogenannten „Byte-Code“, der durch jeden Java-Compiler erzeugt werden kann, nicht direkt auf die Karte ladbar. Dazu muß der Byte-Code, der in einem bestimmten Format – dem „Class-File-Format“ – vorliegt, zunächst in das sogenannte CAP-Format überführt werden. Leider unterscheiden sich die CAP-Formate der verschiedenen JavaCard-Hersteller, so daß die Kartenapplikation nach der Überführung in das CAP-Format eines Kartenherstellers nicht auf die Karte eines anderen Herstellers geladen werden kann.

Derzeit sieht es so aus, als ob sich die unterschiedlichen Hersteller nicht auf ein einheitliches CAP-Format einigen können, da sich im CAP-Format die jeweils verwendete unterschiedliche Hard- und Software widerspiegelt, und daher nicht zu erwarten ist, daß auf dieser Ebenen eine Vereinheitlichung stattfinden wird. Vielmehr wird außerhalb der Karten – in den Kartenladeterminals – eine genormte Infrastruktur zum Laden von Applikationen auf die Karte entstehen, die erkennen kann, auf welchen Kartentyp eine Applikation geladen werden soll, und die dann vor dem Laden das korrekte CAP-Format erstellt. Dies erfordert allerdings nicht nur zusätzliche Software in den Kartenterminals, sondern erhöht auch den Verwaltungsaufwand. Vor diesem Hintergrund ist daher auf absehbare Zeit kaum an eine völlige Interoperabilität zwischen verschiedenen Java-basierten Smartcards zu denken.

4 Fazit

Leicht zu programmierende, multifunktionale Chipkarten stellen für zukünftige verteilte Applikationsszenarien ein interessantes Potential als Teil einer Anwendungsplattform dar. Durch die ihnen eigene Charakteristik der einfachen Anwendbarkeit und leichten Verbreitung bei gleichzeitiger Zusicherung verschiedener Sicherheitseigenschaften sind sie nicht nur als persönlicher und vertrauenswürdiger „Taschenrechner“ einsetzbar, sondern können auch als sichere Ausführungsplattform für Softwarekomponenten in Netzen dienen. Zwar sind sowohl die Hard- als auch die Software noch nicht ausgereizt, es zeichnet sich aber ab, daß die weitere Entwicklung sehr schnell verlaufen könnte. Durch den geplanten Einsatz von multifunktionalen Smartcards im Mobiltelefonbereich, der auch in Deutschland zur Zeit ebenso wie das Internet eine rasante Entwicklung durchläuft, wird die schnelle Verbreitung solcher Karten noch weiter gefördert. Es bleibt allerdings abzuwarten, welche Arten von Applikationen diese neue Plattform in der Praxis hervorbringen wird.

5 Literaturliste

[Bluetooth] http://www.bluetooth.com

[JavaCardAPI] http://java.sun.com/products/javacard/index.html

[Jini] http://java.sun.com/products/jini/

[KIVS99] S. Fünfrocken, F. Mattern*: Mobile Agents as an Architectural Concept for Internet-based Distributed Applications - The WASP Project Approach*, To appear in Proceedings KiVS'99 ("Kommunikation in Verteilten Systemen"), Springer-Verlag, 1999

[Kreft] H. D. Kreft, *Patentanalyse: Indikator für Entwicklungen im Chipkartenmarkt?*, 8. GMD-Smartcard Workshop, Darmstadt 3./4. Feb. 1998

[LY97] T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley, 1997

[Posegga1] J. Posegga, *Java Smart Cards as a Platform for Electronic Commerce*, in: Electronic Commerce - IFIP/GI Working Conference on Trends in Distributed Systems for Electronic Commerce (Hg.: Griffel, Tu, Lamersdorf), dpunkt-Verlag, pp. 175-182, 1998

[Posegga2] J. Posegga, H. Vogt, *Byte Code Verification for Java Smart Cards Based on Model Checking*, in: Proc. 5th European Symposium on Research in Computer Science – ESORICS 98 (Hg.: Quisquater, Deswarte, Meadows, Gollmann), Springer Verlag, LNCS 1485, 1998

[T-Spaces] P. Wyckoff, S. W. McLaughry, T. J. Lehman, D. A. Ford, *T-Spaces*, IBM Journal 37 (3), 454-474, 1998

[Wohlmacher] P. Wohlmacher, *Visionen zur Chipkarte - Gedanken zu Entwicklungen und Anwendungen*, it+ti 5/97, 34-41, 1997