CENTRO UNIVERSITÁRIO FEEVALE

ARIEL CORTES BOESING

PAGAMENTO ELETRÔNICO: ESTUDO TEÓRICO E PROTOTIPAÇÃO DE UM SISTEMA BASEADO EM SMART CARD PARA TRANSAÇÕES PONTO A PONTO EM DISPOSITIVOS MÓVEIS UTILIZANDO BLUETOOTH

Novo Hamburgo, novembro de 2008.

ARIEL CORTES BOESING

PAGAMENTO ELETRÔNICO: ESTUDO TEÓRICO E PROTOTIPAÇÃO DE UM SISTEMA BASEADO EM SMART CARD PARA TRANSAÇÕES PONTO A PONTO EM DISPOSITIVOS MÓVEIS UTILIZANDO BLUETOOTH

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Prof. Ms. Eduardo Leivas Bastos

Novo Hamburgo, novembro de 2008.

RESUMO

O comércio eletrônico pode ser definido como qualquer transação de negócios em que as ações se dão por meio eletrônico, podendo, para tanto, fazer uso de um sistema de pagamento eletrônico. Estes sistemas são viabilizados pela existência de alguma espécie de dinheiro eletrônico e devem utilizar métodos eficientes para garantir a integridade e segurança destes valores, seja no armazenamento ou na sua manipulação. Para suprir essa necessidade, alguns sistemas de pagamento eletrônico, como as carteiras eletrônicas, implementam soluções baseadas em *smart cards*. Estes, na maior parte dos casos, são dispositivos dotados de um *chip* composto por memórias RAM, ROM e EEPROM, além de um processador e portas de entrada e saída, que lhe conferem a capacidade de processamento e armazenamento seguro de informações. Dentre os *smart cards*, existem aqueles utilizados como base para autenticação de assinantes da tecnologia de telecomunicação móvel GSM, chamados de SIM *cards*. Um SIM *card* nada mais é que um *smart card* em tamanho reduzido para ser utilizado em dispositivos móveis como aparelhos celulares. Assim, o presente trabalho teve como objetivo desenvolver um protótipo via simulação de sistema de pagamento eletrônico, do tipo carteira eletrônica, baseado em *smart card*, para dispositivos móveis. O protótipo desenvolvido demonstrou que é possível a criação de aplicações que possibilitem o pagamento eletrônico por meio de transferência de fundos armazenados de forma segura em *smart cards*. Demonstrou, também, que é possível que esses valores sejam transmitidos através de uma comunicação *Bluetooth*, ponto a ponto, entre dois aparelhos celulares simulados padrão GSM, sem o envolvimento da rede de telefonia celular e/ou um servidor de aplicação para validação das transações. Salienta-se que esse projeto não se esgota no presente trabalho, uma vez que um dos maiores desafios dessa arquitetura é garantir que todo o processo ocorra de maneira segura aos usuários e instituições administradoras do sistema. Desta forma, o desenvolvimento da camada de segurança da aplicação, por meio da utilização de certificados digitais e chaves criptográficas de sessão, geradas e administradas dentro do *smart card*, deve ser questão abordada em trabalhos futuros. Ainda, o aprofundamento dessa pesquisa se torna atrativo, tendo em vista que a mobilidade para aplicações de pagamento eletrônico e demais soluções do gênero, ou seja, a concentração de serviços em dispositivos móveis, é apontada por especialistas como uma tendência mundial que já vem se consolidando.

Palavras-chave: Pagamento eletrônico. *Smart card*. *Java Card*. *Bluetooth*. Dispositivos Móveis.

ABSTRACT

The e-commerce can be defined as any business transaction whose actions occur through an electronic way that can make use of an electronic payment system. These systems are enabled by the existence of any type of electronic money and must use efficient methods to ensure the integrity and security of these values, either in storage or in its handling. To supply these requirements, some electronic payment systems, such as electronic purses, implement solutions based on smart cards. In most cases, these cards are devices equipped with a chip composed of memories RAM, ROM and EEPROM, a processor and I/O ports, which give the ability of secure process and secure storage information. Among the smart cards, there are those used as a basis for authentication of subscribers on GSM mobile telecommunication technology, called SIM cards. A SIM card is nothing more than a smart card reduced in size to be used in mobile devices, such as cell phones. Therefore, this study aimed to develop an electronic payment prototype simulation system, like the electronic purse kind, based on smart card for mobile devices. The prototype developed has shown it is possible to create applications that allow the payment through electronic transfer of funds stored safely in smart cards. It also demonstrated that it is possible to transmit these values through a Bluetooth communication, peer-to-peer, between two simulated standard GSM mobile devices, without the involvement of the mobile cell phone network and/or a server application to validate the transaction. It can be noted that this project does not end in this work, since one of the biggest challenges of this architecture is to ensure that the whole process occurs in a secure manner to users and system's administrator institutions. Thus, the development of the application security layer based on digital certificates and cryptographic keys session, created and administered within the smart card, should be discussed in future work. Still, the deepening of this research becomes attractive once that mobility for electronic payment applications and other similar solutions, that is, the concentration of services on mobile devices, is identified by experts as a worldwide trend that has been already consolidating.

Key words: Eletronic Payment. Smart card. Java Card. Bluetooth. Mobile Device.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3DES	Triple Data Encryption Standard
3GPP	3rd Generation Partenership Project
AES	Advanced Encryption Standard
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ATM	Automated Teller Machine
ATR	Answer to Reset
BNEP	Bluetooth Network Encapsulation Protocol
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAS	Digital Signature Algorithm
ECC	Elliptic Curve Cryptosystems
EDGE	Enhanced Data rates for Global Evolution
EEPROM	Electrically Erasable Programmable Read Only Memory
GAP	Generic Access Profile
GCF	Generic Connection Framework
GOEP	Generic Object Exchange Profile
GPRS	General Packet Radio Service
GSM	Global System for Mobile
HCI	Host Controller Interface
HSDPA	High-Speed Downlink Packet Access

HSPA	High-Speed Packet Access
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
J2ME	Java Micro Edition
JCP	Java Community Process
JCRE	Java Card Runtime Enviroment
JCVM	Java Card Virtual Machine
JSR	Java Specification Requests
JTWI	Java Technology for the Wireless Industry
LED	Light Emitting Diode
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
MSA	Mobile Service Architecture
NPU	Numerical Processing Unit
OBEX	Object Protocol Exchange
PAN	Personal Area Network
PC/SC	Personal Computers / Smart Card
PDA	Personal Digital Assistant
PKI	Public Key Infra-Estructure
POS	Point of Sale
RAM	Random Access Memory
RF	Radio Freqüência
ROM	Read Only Memory
RSA	Rivest, Shamir and Adleman
SCQL	Structured Card Query Language
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol
SIM	Subscriber Identity Module
S/MIME	Secure / Multipurpose Internet Mail Extensions
SMS	Short Message Service
SPP	Serial Port Profile
SSCD	Secure Signature Creation Device
TCS	Telephony Control Protocol Specification

TDES	Triple Data Encryption Standard
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver Transmitter
UMTS	Universal Mobile Telecommunications System
USAT	USIM Application Toolkit
USIM	Universal Subscriber Identity Module
WAP	Wireless Application Protocol
WCDMA	Wideband Code Division Multiple Access
WIM	Wireless Identity Module
WML	Wireless Mark-up Language
WTK	Wireless Toolkit

SUMÁRIO

INTRODUÇÃO

O comércio eletrônico pode ser definido como "qualquer forma de transação de negócios em que as ações se dão por meio eletrônico" (FERREIRA, 1999, p. 4), seja por meio de transferência eletrônica de documentos, Internet, cartões de crédito, *smart cards*, aparelhos celulares, PDAs ou televisão digital. Muitas operações realizadas através de comércio eletrônico fazem uso de algum sistema de pagamento eletrônico e dinheiro eletrônico para viabilizar as compensações monetárias pela compra de bens ou serviços.

Em termos gerais, dinheiro eletrônico pode ser definido como um valor monetário armazenado em um dispositivo eletrônico capaz de emitir e receber esses valores como forma de pagamento (CARAT, *apud* BADDELEY, 2004, p. 240). Segundo o *Europe Central Bank* (ECB, apud BADDELEY, 2004, p. 240), o dinheiro eletrônico pode ser definido como o armazenamento eletrônico de valores em um dispositivo específico que pode ser usado para realizar pagamentos a empresas, que não seja o próprio emitente, sem necessariamente envolver contas bancárias nas transações, atuando com um instrumento pré-pago ao portador.

Atualmente, os esforços dos fornecedores de soluções de pagamento eletrônico têm se concentrado em sistemas de vendas pela Internet, sistemas de *Mobile Payment*[1] e sistemas de *e-Purse* (carteira eletrônica). Esse novo conceito de pagamento, chamado de pagamento eletrônico, traz consigo o desafio de tornar-se confiável e acessível a um grupo cada vez maior de usuários, sendo parte integrante da rotina diária dos consumidores.

Dentre as diversas soluções para sistemas de pagamento eletrônico encontram-se aquelas baseadas em *smart cards*. Segundo Lorenzoni (2006, p. 55), "a maioria dos mais promissores tipos de dinheiro eletrônico e sistemas de pagamento são baseados em *smart*

[1] Sistemas que possibilitam a realização de pagamentos através de dispositivos móveis como aparelhos celulares.

cards". Lorenzoni (2006, p. 55) complementa afirmando que "esses sistemas de compra, baseados em *smart cards*, contam com segurança, confiabilidade e soluções de baixo custo, sem contar no modo atrativo com que substituem o manejo de dinheiro em espécie".

Para Rankl e Effing (2003, pág. 673), *smart cards* são, pela sua natureza particular, adequados aos sistemas de pagamento eletrônico. Eles são dispositivos com dimensões semelhantes às de um cartão de crédito, dotados de um *chip* de cerca de 1,4 centímetros de largura e 1,2 centímetros de comprimento, compostos por memórias ROM e EEPROM (*Memory cards*), podendo também possuir uma memória RAM, além de um microprocessador e um co-processador matemático e portas de entrada e saída, que lhe conferem a capacidade de processamento e armazenamento de informações de forma fácil e segura (*Microprocessor cards*). Suas especificações técnicas são padronizadas pela família de normas ISO/IEC 7816. Conforme Rankl e Effing (2003, p. 673), "desde que os *smart cards* passaram a realizar ativamente cálculos complexos sem serem influenciados por fatores externos foi possível desenvolver novas abordagens para operações de pagamentos". Um exemplo disso são as carteiras eletrônicas, que só se tornaram possíveis através deste meio.

Em um sistema de carteira eletrônica, uma quantia de dinheiro eletrônico é armazenada antes da realização de qualquer pagamento. Quando um pagamento é realizado ocorre uma transferência de valores entre os pares, creditando o vendedor ao mesmo tempo em que o comprador tem seu saldo debitado, nos mesmos valores. Posteriormente o vendedor pode solicitar o reembolso dos valores recebidos eletronicamente para moeda em espécie junto ao operador do sistema de pagamento. Analogicamente, um comprador supre sua carteira eletrônica através da troca de moeda em espécie por dinheiro eletrônico, também junto ao operador do sistema.

Assim, sistemas de carteira eletrônica proporcionam benefícios significativos às partes envolvidas, seja pela redução de custos vinculados ao gerenciamento de dinheiro em espécie, pela eliminação de qualquer custo proveniente da comunicação em cada transação, uma vez que as transferências ocorrem *off-line*, pela redução do risco associado a roubos ou pela comodidade e facilidade de uso, principalmente se a carteira eletrônica for disponibilizada através de dispositivos móveis como aparelhos celulares.

Originalmente, as aplicações baseadas em *smart cards* estavam voltadas para o setor de telecomunicações. Entretanto, recentemente os *smart cards* tem se posicionado em outros setores de mercado, como os sistemas de pagamento eletrônico. Em outro momento, soluções baseadas em *smart card* começaram a ser utilizadas na área de telefonia móvel no padrão

GSM. A tecnologia GSM baseia-se na utilização de *chips* conhecidos como SIM *cards*. Por sua vez, um SIM *card* nada mais é que um *smart card* em proporções menores e específicas. Segundo dados da *Wireless Intelligence*[2], publicados no *site* da *GSM Association*[3], ao final do ano de 2007, o mundo contava com mais de 2,6 bilhões de aparelhos celulares equipados com a tecnologia GSM. Considerando que cada aparelho GSM possui um chip do tipo *smart card* e que a maioria dos modelos atuais tem capacidade para executar programas escritos na linguagem Java, percebe-se aí um mercado potencial para aplicações móveis de pagamento eletrônico.

Para que exista a interação necessária, um sistema de pagamento eletrônico do tipo carteira eletrônica deve estabelecer um protocolo que possibilite a troca e o gerenciamento de mensagens entre os pares da aplicação. Para tanto, o sistema deve prover um meio de comunicação entre os dispositivos participantes da transação, seja por contato físico ou alguma tecnologia de comunicação sem fio. No caso de dispositivos móveis, como aparelhos celulares, essa comunicação pode ser realizada através da rede da empresa de telefonia, realizando transações *on-line* através de um servidor de aplicação, ou por meio de outras tecnologias sem fio como as especificações *Bluetooth* ou *Infrared*[4], realizando transações *off-line*, processadas no *smart card*, através da comunicação ponto a ponto entre os pares.

Segundo Thompson, Kline e Kumar (2008, p. 3), a tecnologia sem fios *Bluetooth* é uma especificação aberta de baixo custo, baixa potência e que faz uso de tecnologia de rádio de curto alcance para comunicação sem fio *ad hoc*[5] de voz e dados, em qualquer lugar do mundo. Ainda, a especificação *Bluetooth* define padrões para assegurar a compatibilidade de dispositivos de diferentes fornecedores. Assim, considerando as particularidades de um sistema de carteira eletrônica em dispositivos móveis, a tecnologia *Bluetooth*, juntamente com a plataforma J2ME e as atuais APIs (*Application Programming Interface*) existentes, se mostra adequada para prover o meio de comunicação necessário.

No entanto, dentre os modelos conhecidos de carteira eletrônica em dispositivos móveis, sejam cartões ou aparelhos celulares, estão aqueles que ou utilizam transações *on-line*, com a necessidade de autenticação das partes junto a um servidor de aplicação da empresa operadora do serviço fazendo uso de alguma rede de telecomunições, ou utilizam um

[2] https://www.wirelessintelligence.com
[3] http://www.gsm.org
[4] Tecnologia de conexão sem fio que emprega raios infravermelhos como forma de comunicação.
[5] Rede que não possui um nó ou terminal especial para o qual todas as comunicações convergem, de forma que um usuário se comunica diretamente com outro(s). Não há topologia predeterminada, nem controle centralizado.

hardware especializado, uma espécie de POS, para que os valores sejam transmitidos entre os *smart cards* através do contato físico entre os cartões (vendedor e comprador) e as leitoras deste hardware. Percebeu-se, então, a possibilidade de se propor um modelo de carteira eletrônica, baseado nos elementos de segurança de um SIM *card*, cujas transações se dêem de forma *off-line* através da comunicação *Bluetooth* entre dois dispositivos móveis. Tal possibilidade leva, então, a seguinte questão de pesquisa: **como funcionaria um sistema de carteira eletrônica com essas características**?

Assim, dado o exposto anteriormente, a resposta para a questão acima se torna a motivação para o presente trabalho e leva ao estabelecimento dos seguintes objetivos gerais e específicos a serem atingidos:

Objetivo geral

Projetar e desenvolver um protótipo de sistema de pagamento eletrônico baseado em *smart cards* para dispositivos móveis. Este sistema deve permitir a transferência de dinheiro eletrônico através da comunicação ponto a ponto, via *Bluetooth*, entre dois aparelhos celulares.

Objetivos específicos

a) Estudar e apresentar os conceitos associados a sistemas de pagamento eletrônico e carteira eletrônica;

b) Identificar e apresentar os principais conceitos acerca das tecnologias *Smart Card*, *Java Card* e *Bluetooth*;

c) Estudar ferramentas, *frameworks* e ambientes de desenvolvimento e simulação de aplicações J2ME, *Java Card*, *Smart Card* e *Bluetooth*;

O conteúdo do presente trabalho está organizado e distribuído em seis capítulos, além da introdução e conclusão. A introdução contextualiza o tema estudado e os objetivos propostos ao projeto, o capítulo 1 aborda os conceitos de pagamento eletrônico e dinheiro eletrônico, o capítulo 2 explora o referencial teórico acerca da tecnologia *Smart Card*, o capítulo 3 aborda a plataforma *Java Card*, o capítulo 4 traz conceitos que permeiam a especificação *Bluetooth*, o capítulo 5 apresenta a descrição e contextualização do protótipo proposto, bem como a metodologia e ferramentas utilizadas para desenvolvimento, teste e simulação, o capítulo 6 apresenta os resultados alcançados após as etapas de implementação e

teste do protótipo, além de limitações e trabalhos futuros para o projeto, e, por fim, são apresentadas as conclusões do presente trabalho.

1 PAGAMENTO ELETRÔNICO

Neste capítulo são apresentados conceitos a respeito do tema estudado e que servem de embasamento teórico para este projeto. Assim, a seguir são abordados os conceitos sobre sistemas de pagamento eletrônico, dinheiro eletrônico e sistemas de pagamento eletrônicos com *smart cards*.

1.1 Sistemas de Pagamento Eletrônico

O advento das soluções de comércio eletrônico trouxe consigo diversas soluções de pagamento eletrônico que visam permitir o pagamento pela compra de bens ou serviços. Estes sistemas normalmente fazem uso de tecnologias de informação, tecnologias de comunicação, como a Internet, redes de operadores de telefonia e redes *wireless*, e dispositivos eletrônicos, como computadores, telefones celulares, PDAs e, futuramente, televisores digitais.

Um sistema de pagamento eletrônico pode ser definido como um conjunto de tecnologias, regras, protocolos e costumes que tornam possível que empresas e pessoas troquem dinheiro eletrônico uma com as outras. Da mesma forma que os sistemas de pagamentos atuais constituem um dos principais pilares da economia moderna, os sistemas de pagamento eletrônico constituirão um importante pilar para as economias do futuro.

Conforme Kniberg (2002, pág. 10), um sistema de pagamento eletrônico deve responder algumas questões referentes ao dinheiro eletrônico do qual fazem uso, como por exemplo: de que maneira um usuário sabe quanto dinheiro possui? Onde o dinheiro eletrônico é armazenado? Como ele pode ser acessado? Como o sistema sabe quem é o proprietário de determinados valores? De que forma um dinheiro muda de propriedade? Como podem ser realizadas trocas de dinheiro eletrônico entre diferentes sistemas? As operações são passíveis de rastreamento?

O sucesso de um sistema de pagamento eletrônico está associado as suas capacidades de ser seguro, de garantir a sua integridade e de suas transações, de ter uma ampla aceitação e difusão em território nacional, de ser de fácil uso, ter um baixo custo associado a suas transações e, ainda, transmitir credibilidade e confiança, além de permitir pagamentos dentro de uma grande faixa de valores e sempre proporcionar agilidade nas suas operações (KNIBERG, 2002, pág. 14).

Ainda, sistemas de pagamento eletrônico devem garantir a atomicidade de suas transações e, principalmente, a segurança da sua infra-estrutura, de maneira que não sejam possíveis falsificações ou adulterações através da interceptação dos dados de uma transação.

Dentre as características de um sistema de pagamento eletrônico, o anonimato é a que desperta diferentes opiniões, dependendo do contexto em que é aplicado. Operações fraudulentas e ilegais somente terão seus responsáveis identificados se permitirem a sua rastreabilidade. Entretanto, um usuário honesto pode querer manter a sua transação anônima da mesma forma que a realiza quando se faz presente pessoalmente em uma loja qualquer e realiza o pagamento utilizando dinheiro em espécie.

Para Rankl e Effing (2003, pág. 673), dadas as necessidades dos sistemas de pagamento eletrônico, dispositivos como *smart cards* tornam-se, pela sua natureza particular, adequados a eles. *Smart cards* fornecem um meio seguro, robusto, amigável e de fácil uso, além de serem capazes de realizar cálculos criptográficos complexos sem a influência de fatores externos ao seu hardware, garantindo, assim a segura efetivação de transações financeiras.

Atualmente os sistemas de pagamento eletrônico têm sido impulsionados, principalmente, por bancos que buscam expandir seus canais de auto-atendimento. O *mobile banking*, ou *m-banking*, tem consumido esforços dos principais bancos atuantes no Brasil para o desenvolvimento de aplicações bancárias para aparelhos celulares. O resultado é que hoje os clientes desses bancos já podem realizar transações bancárias que vão desde consultas de saldo até aplicações financeiras. Alguns bancos, em menor número, oferecem o serviço de pagamentos de bens e serviços utilizando o celular, como o produto Banricompras Celular do banco Banrisul. Estes produtos atuam como cartões de débito através de transações autenticadas de forma *on-line* junto aos bancos.

1.2 Dinheiro Eletrônico

Diversos autores definem dinheiro eletrônico como sendo o armazenamento eletrônico de valores em dispositivos específicos e que podem ser usados para realizar pagamentos a empresas, que não seja o próprio emitente, sem necessariamente envolver contas bancárias nas transações, atuando com um instrumento pré-pago ao portador.

O dinheiro eletrônico surge como uma tendência para um futuro próximo. Na visão de Pan (2005), o dinheiro eletrônico é a segunda transformação radical nas formas monetárias e impactará não somente no comércio eletrônico, mas também no sistema monetário atual e suas políticas. Assim, chegará o momento em que os Bancos Centrais deverão analisar cuidadosamente o desenvolvimento das moedas eletrônicas, estabelecendo adequadas políticas de gestão econômicas que considerem rigorosamente o risco associado a este meio.

Para que possa ser utilizado com a mesma flexibilidade que dinheiro em espécie, o dinheiro eletrônico, ou moeda eletrônica, deve possuir algumas características essenciais, as quais, se não estiverem presentes, podem comprometer ou limitar as capacidades da moeda eletrônica.

Para Rankl e Effing (2003, pág. 679), as propriedades essenciais que moedas eletrônicas devem possuir para minimizar as suas diferenças com dinheiro real são as seguintes:

a) **Processamento** - um ponto importante, embora, em princípio trivial, é a propriedade que as moedas eletrônicas devem possuir de serem completa e automaticamente processadas em máquinas;

b) **Transferências** – moedas eletrônicas não devem estar vinculadas a um único meio de transferência e armazenamento, como *smart cards*;

c) **Divisibilidade** – moedas eletrônicas devem ser divisíveis de forma que se possa pagar qualquer quantia desejada sem a necessidade de recorrer a dinheiro real;

d) **Descentralização** – diferentemente do que ocorre com as transações de cartão de crédito, sistemas que utilizam dinheiro eletrônico devem ser capazes de operar de forma descentralizada, realizando transações *off-line* sem a necessidade de autenticação junto a um servidor de aplicação. Esta descentralização deve ser análoga a transferência de dinheiro real de mão em mão;

e) **Transferibilidade** – moedas eletrônicas devem possuir a propriedade de transferir-se diretamente de um usuário para outro, como ocorre com o dinheiro real quando transferido da carteira do comprador para as mãos do vendedor, sem a necessidade de uma terceira entidade envolvida para validação da transação;

f) **Monitoração** – apesar da exigência do anonimato, o dinheiro eletrônico deve ser passível de monitoração pela entidade responsável pelo sistema, uma vez que essa é a única forma de reconhecer e eliminar fraudes e falhas de segurança. A entidade operadora do sistema deve ser responsável por acompanhar e garantir a consistência dos fluxos de pagamentos;

g) **Segurança** – é uma propriedade fundamental de qualquer sistema que faça uso de dinheiro eletrônico. Qualquer sistema rapidamente entrará em colapso caso seja possível forjar ou duplicar moedas eletrônicas. Esta é a razão pelo qual é indispensável a utilização de funções criptográficos fortes, pois somente assim é possível atingir o nível de segurança necessário;

h) **Aceitação** – o dinheiro eletrônico deve ser legalizado e aceito em todo o território nacional do país de origem.

Schmees (2003, pág. 136), complementando as idéias apresentadas por Rankl e Effing, diz que o dinheiro digital não pode expirar, e deve ser válido até que o dispositivo em que estão inseridas seja destruído. Ainda, as moedas eletrônicas devem poder ser usadas a partir de qualquer participante e podem ser trocados em qualquer direção dentro de uma transação comercial.

1.3 Sistemas de Pagamento Eletrônico com *Smart Cards*

Existem três modelos fundamentais de pagamento eletrônico com a utilização de *smart cards*: cartões de crédito (*pay later*) - o pagamento é realizado depois do serviço prestado, cartões de débito (*pay now*) - o pagamento é realizado no momento em que o serviço é prestado - e cartões de carteira eletrônica (*pay before*) - o pagamento é realizado antes do serviço ser realizado. A figura 1.1 ilustra a classificação dos *smart cards* utilizados em pagamentos eletrônicos.

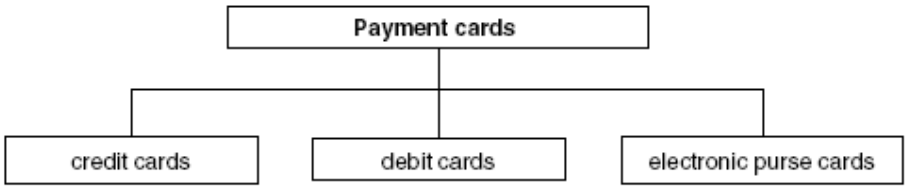

Figura 1.1 – Classificação dos *smart cards* utilizados em pagamentos eletrônicos.
Fonte: RANKL E EFFING (2003, pág. 674).

1.3.1 Carteiras eletrônicas (*e-purses*)

Carteiras eletrônicas podem ser definidas como dispositivos eletrônicos capazes de armazenar dinheiro eletrônico e permitir o pagamento de bens e serviços através de transações seguras entre duas partes.

A idéia de implementar uma carteira eletrônica em um *smart card* remonta aos primórdios dessa tecnologia. No entanto, só em meados da década de 1990 este conceito começou a ser aplicado com o desenvolvimento dos primeiros sistemas de carteira eletrônica (RANKL e EFFING, 2003, pág. 685).

Uma carteira eletrônica deve atuar como um instrumento pré-pago ao portador, de forma semelhante como acontece com os cartões telefônicos pré-pagos. Assim, uma quantia em dinheiro eletrônico deve ser armazenada antes da realização de qualquer pagamento. Quando um pagamento é realizado ocorre uma transferência de valores entre comprador e vendedor, creditando o vendedor ao mesmo tempo em que o comprador tem seu saldo debitado, nos mesmos valores. Posteriormente o vendedor pode solicitar o reembolso dos valores recebidos eletronicamente para moeda em espécie junto ao operador do sistema de pagamento. Analogicamente, um comprador supre sua carteira eletrônica através da troca de moeda em espécie por dinheiro eletrônico, também junto ao operador do sistema.

Com exceção do procedimento de troca de dinheiro em espécie por dinheiro eletrônico, e vice-versa, todas as demais etapas de uma transação devem ser realizadas de forma *off-line*, sem o envolvimento de uma terceira parte para realizar a validação da operação. As transações devem ser realizadas apenas através do contato ponto a ponto entre as partes envolvidas (comprador e vendedor).

Assim, sistemas de carteira eletrônica proporcionam benefícios significativos às partes envolvidas, seja pela redução de custos vinculados ao gerenciamento de dinheiro em

espécie, pela eliminação de qualquer custo proveniente da comunicação em cada transação, uma vez que as transferências ocorrem *off-line*, pela redução do risco associado a roubos ou pela comodidade e facilidade de uso, principalmente se a carteira eletrônica for disponibilizada através de dispositivos móveis como aparelhos celulares.

Alguns bancos, como o Unibanco e Bradesco, disponibilizam sistemas denominados por eles como Carteira Eletrônica. Porém, na verdade esses sistemas não armazenam dinheiro eletrônico, mas sim dados de cartões de crédito e débito de forma eletrônica, fugindo do conceito original de carteira eletrônica. São sistemas voltados principalmente para compras na Internet que atuam de forma *on-line*, autenticando as compras junto a um servidor de aplicação.

A condição fundamental para alcançar a compatibilidade entre diversos sistemas de carteira eletrônica é a existência de um documento especificando as diversas características comuns que esses sistemas devem ter. Assim, em 1999 a CEPSCO publicou a primeira versão da "*Common Electronic Purse Specifications*" (CEPS).

A CEPS apresenta funções padrões para sistemas de carteira eletrônica, seja para transações *off-line* como transações *on-line*. Baseia-se na norma européia para carteiras eletrônicas, a EN 1546, porém, apresentando várias extensões e modificações em relação a esta norma. Por exemplo, ao contrário do EN 1546, ela recomenda a utilização de certificados para a autenticação de terminais e *smart cards*, além do algoritmo 3-DES para funções criptográficas. Essa especificação é especialmente otimizada para uso em dispositivos como *smart card* (RANKL e EFFING, 2003, pág. 702).

Seguindo o pioneirismo europeu em sistemas de carteira eletrônica, ainda na década de 90 surgiram os sistemas *Proton* e o *Mondex System*, ambos baseados na tecnologia *smart card*.

2 SMART CARDS

A seguir são apresentados os principais conceitos acerca da tecnologia *Smart Card* e que servem de embasamento teórico para o desenvolvimento do protótipo alvo deste trabalho. São abordados tópicos que passam pela visão geral da tecnologia *Smart Card*, a história dos *smart cards*, sua arquitetura e componentes, elementos de segurança, normas técnicas associadas, bem como uma visão sobre os SIM *cards* (*Subscriber Identity Module Card*) e sua arquitetura lógica.

2.1 *Smart cards* – Visão Geral

Segundo a *Smart Card Alliance*[6], *smart cards* podem ser definidos como dispositivos que possuem um circuito integrado embutido composto por um microcontrolador seguro, ou com inteligência equivalente, e memória interna ou ainda serem somente um *chip* de memória, podendo conectar-se a uma leitora ou terminal através de contato físico direto ou uma interface de rádio freqüência. Ainda, conforme a *Smart Card Alliance*, o fato de possuírem um microcontrolador embutido traz aos *smart cards* a capacidade de armazenarem uma quantidade considerável de informações e realizar funções de criptografia e autenticação mútua no próprio cartão através da interação com uma leitora de *smart cards*.

Para Ortiz (2003a), *smart card* é um cartão de plástico que contém um circuito integrado embutido, sendo, por concepção, altamente seguro. Quando no formato ID-1 um *smart card* é muito semelhante a um cartão de crédito. Entretanto, quando usado como um SIM *card* (formato ID-000), o cartão de plástico é pequeno, somente grande o suficiente para caber em um telefone celular. Segundo Savoldi e Gubian (2007, pág. 182), todo SIM/USIM *card* é um *smart card* de contato padronizado pela família de normas ISO/IEC 7816.

[6] http://www.smartcardalliance.org/

Na visão de Rankl e Effing (2003, pág. 19), uma das mais importantes vantagens da tecnologia *smart card* é que os dados podem ser armazenados de forma segura, protegidos contra acesso e manipulação não autorizados. Através de recursos de hardware e software são implementados métodos para que os dados confidenciais do cartão não possam ser manipulados por algum meio externo ao *chip*, garantindo, portanto, que as informações somente sejam lidas, escritas ou apagadas pela unidade de processamento do *smart card*.

Conforme Ortiz (2003a), os *smart cards* mais avançados são equipados com processador e memória e podem ser usados em aplicações seguras baseadas em infra-estrutura de chaves públicas ou algoritmos de chaves compartilhadas. Ainda, para Ortiz (2003a), a memória não volátil de um *smart card* é o seu recurso mais precioso e pode ser utilizado para armazenamento de chaves secretas e certificados digitais. Para auxiliar no processamento de algoritmos criptográficos, alguns *smart cards* possuem co-processadores que suportam algoritmos como RSA, AEC, DES e 3DES.

Assim, devido a essas capacidades os *smart cards* se tornaram convenientes e amigáveis módulos de segurança para processamento e armazenado de informações. Para Rankl e Effing (2003, pág. 673), *smart cards* são, pela sua natureza particular, adequados aos sistemas de pagamento eletrônico, pois suas características proporcionam um ambiente seguro ao armazenamento e manipulação de informações críticas.

Atualmente, apesar de serem empregados em diversas setores e aplicações, o uso de *smart cards* tem sido impulsionado principalmente pelo setor bancário e de telecomunicações.

2.2 A história dos *Smart cards*

O desenvolvimento da tecnologia *Smart Card* tem origem na década de 1970, com o progresso das tecnologias que permitiam integrar o armazenamento e processamento de dados em um único chip de silício medindo alguns milímetros quadrados (RANKL e EFFING, 2003, pág. 3).

Patentes utilizando estes circuitos em cartões de identificação foram registradas em 1968 e 1970. Porém, foi em 1974 que surgiram os verdadeiros progressos, quando Roland Moreno registrou sua patente de *smart card* na França. Durante o progresso da tecnologia, os principais avanços vieram de pesquisas originadas na Alemanha e na França, fazendo com

que esses países desempenhassem papel líder no desenvolvimento e comercialização de *smart cards* (RANKL e EFFING, 2003, pág. 3).

O grande avanço da tecnologia *smart card* foi dado em 1984 quando a Agência de Serviços Postais e de Telecomunicações Francesa obteve êxito em seus experimentos com *smart cards*. A pesquisa acabou provando que estes cartões propiciavam alta confiabilidade e proteção contra a manipulação das informações ali armazenadas (RANKL e EFFING, 2003, pág. 4).

Os c*hips* microprocessados, que são significativamente maiores e mais complexos que os *memory cards*, foram utilizados pela primeira vez em larga escala na área de telecomunicações. Em 1988, o *German Post Office* introduziu um cartão microprocessado contendo uma memória EEPROM como um cartão de autorização para a rede de telefonia móvel analógica (C-Netz). A motivação para a introdução dessa nova tecnologia foi o aumento do número de fraudes em cartões com tarja utilizados até o momento. A experiência positiva adquirida pelo *German Post Office* foi decisiva para a utilização de *smart cards* nas redes GSM (RANKL e EFFING, 2003, pág. 4). A rede GSM entrou em operação em 1992 em vários países europeus e se expandiu alcançando, ao final do ano de 2007, mais de 2,6 bilhões de aparelhos celulares em paises de todo o mundo, segundo dados da *Wireless Intelligence*, publicados no *site* da *GSM Association*.

A partir daí, os *smart cards* provaram ser um meio ideal para realizar serviços de autenticação e armazenamento seguro, pois eram capazes de armazenar informações e chaves secretas com elevado nível de segurança e ainda e executar algoritmos criptográficos complexos. Aliado as suas características técnicas estava o tamanho e o formato físico dos cartões, pois se tratavam de dispositivos pequenos e amigáveis, de simples manuseio e fácil adaptação ao cotidiano dos seus usuários, podendo substituir facilmente os, então, atuais cartões com tarja magnética.

Outro importante marco para o progresso dos *smart cards* ocorreu em 1994 com a publicação da primeira versão da especificação EMV. Esta especificação foi o resultado do esforço das três maiores organizações de cartões de crédito do mundo naquele momento, Europay, Master*card* e Visa, para padronizar e garantir a compatibilidade dos *smart cards* no futuro.

Já em 1996, cartões com funções de POS e carteira eletrônica foram emitidos por toda a Áustria, fazendo com que se tornassem o primeiro país no mundo a possuir um sistema de moeda eletrônica (RANKL e EFFING, 2003, pág. 5).

No Brasil o uso de *smart cards* teve seu maior impulso com as redes de telefonia GSM. Aos poucos os bancos e operadoras de cartão de crédito também tem migrado seus atuais cartões com tarja magnética para cartões com chip, devido, principalmente, a grande quantidade de fraudes relacionadas à clonagem de cartões. Um exemplo de sucesso de aplicação *smart card* é o banco Banrisul que, em 2005, disponibilizou aos seus clientes o Cartão Internet Banrisul, que nada mais é do que um *smart card* para acesso aos canais de atendimento *Home* e *Office Banking*. Esta experiência trouxe um grande incremento de segurança às transações na Internet sendo reconhecido e premiado pela mídia especializada. Para fazer uso dessa tecnologia o cliente se utiliza de uma leitora PC/SC conectada a porta USB de seu computador.

Outro setor que tem impulsionado o uso de *smart cards* no Brasil é o setor público, por meio das funções de *e*-Government, possíveis através do uso do *e*-CPF e *e*-CNPJ.

2.3 Classificação dos cartões

Os diversos tipos de cartões existentes podem ser classificados hierarquicamente como demonstrado na figura 2.1 a seguir.

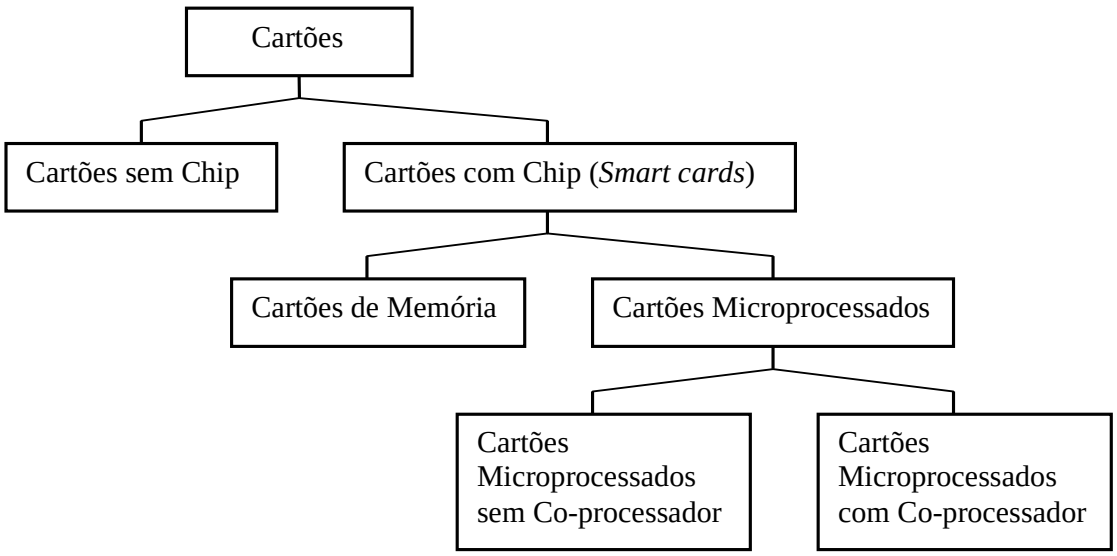

Figura 2.1 – Classificação dos cartões com e sem *chip*.
Fonte: Adaptado de RANKL (2007, pág. 1).

Como ilustrado na figura 2.1, os cartões podem ser divididos em dois grandes grupos: cartões com *chip* e cartões sem *chip*. Os cartões com *chip* podem possuir *chips* mais simples, chamados de *chips* de memória, ou *chips* com microprocessador. Estes últimos, por sua vez, podem possuir *chips* microprocessados com ou sem co-processador para a execução de algoritmos criptográficos como o RSA e/ou o ECC (RANKL, 2007, pág. 2).

2.3.1 Cartões de memória (*memory cards*)

Os primeiros *smart cards* utilizados em grande escala foram os cartões de memória para aplicações de serviços telefônicos (RANKL e EFFING, 2003, pág. 6). Eram cartões pré-pagos que armazenavam valores de forma eletrônica no seu *chip* e os decrementava a medida que os serviços de telefonia eram utilizados. Estes cartões, diferentemente dos cartões que utilizam tarja magnética, impediam que os usuários incrementassem seus créditos através da

técnica conhecida como "*buffering*", que consiste em armazenar os dados da tarja magnética antes de utilizar um determinado serviço e então, após o uso, reescrevê-la com as informações armazenadas anteriormente, recuperando o estado anterior do cartão.

Nesses cartões, a memória EEPROM provê a área de armazenamento para leitura e escrita de dados além de armazenarem um número único de identificação do *chip*, enquanto a memória ROM contém informações sobre o tipo do *chip* (RANKL, 2007, pág. 6). O acesso à memória é controlado pela segurança lógica, que no caso mais simples consiste somente em proteção de escrita ou exclusão na memória ou parte dela. Contudo, também existem *chips* de memória com recursos de segurança lógica mais complexos capazes de executarem encriptações simples (RANKL e EFFING, 2003, pág. 19).

A figura 2.2 ilustra a estrutura típica de um *smart card* de memória e contato (*memory card*) com mecanismo de segurança lógica. As setas indicam o fluxo básico de dados e energia dentro do *chip*.

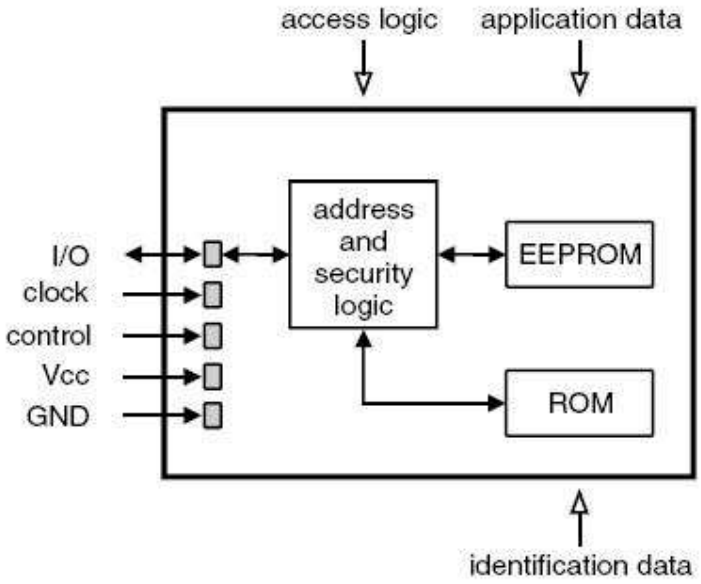

Figura 2.2 – Estrutura típica de um *smart card* de memória.
Fonte: RANKL e EFFING (2003, pág. 19).

A vantagem dos cartões de memória está em seu baixo custo de produção. Em contrapartida, sua aplicação é bastante restrita em comparação aos *smart cards* microprocessados, além de terem de ser descartados após serem totalmente utilizados. Conforme Rankl e Effing (2003, pág. 6), apesar dos *smart cards* de memória ter

funcionalidades limitadas, a sua segurança lógica torna possível proteger dados contra manipulação.

O uso dos cartões de memória não está restrito às aplicações telefônicas, mas em qualquer área que possibilite a utilização de serviços pré-pagos, como estacionamentos, máquinas de refrigerantes, entradas em clubes, transporte público, cafeterias, ou demais sistemas onde o baixo custo de implementação seja primordial.

2.3.2 Cartões com co-processador (*microprocessor cards*)

Para Rankl e Effing (2003, pág. 6), os cartões microprocessados e que possuem um co-processador possibilitaram o aumento do leque de aplicações para os *smart cards*. Com capacidade de armazenar chaves criptográficas e executar algoritmos criptográficos complexos de forma segura, foi, então, possível a implementação dos primeiros sistemas de pagamento *off-line*. A única restrição para o desenvolvimento de novas aplicações para *smart cards* co-processados estava na quantidade pequena e restrita de armazenamento e processamento desses dispositivos.

A figura 2.3 ilustra uma estrutura típica de um *smart card* de contato microprocessado e com co-processador. As setas indicam o fluxo básico de dados e energia dentro do *chip*.

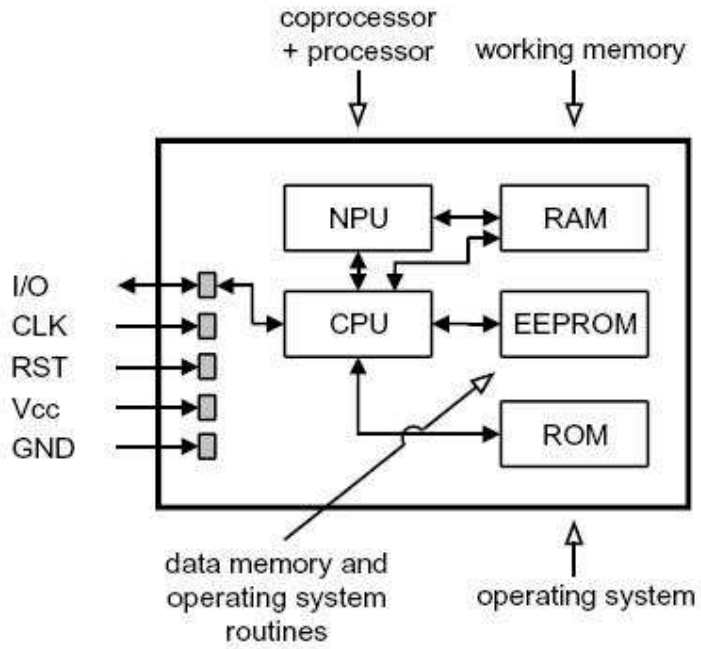

Figura 2.3 – Estrutura básica de um *smart card* microprocessado com co-processador.
Fonte: RANKL e EFFING (2003, pág. 20).

Dentre as áreas de aplicação dos *smart cards* estão as de identificação, controle de acesso à áreas restritas e sistemas informatizados, armazenamento seguro, assinatura eletrônica e carteira eletrônica. Muitos *smart cards* permitem também a existência de diversas aplicações em um único cartão. Ainda, sistemas operacionais modernos, como o *Java Card* ou o Multos, permitem que novas aplicações sejam instaladas um *smart card* mesmo após o cartão já ter sido emitido para o usuário final, com a garantia de não comprometer a segurança das diversas aplicações já existentes no *chip*.

Segundo Rankl e Effing (2003, pág. 7), as principais vantagens dos cartões microprocessados estão na maior capacidade de armazenar dados confidenciais de forma segura e a na capacidade de executar algoritmos criptográficos complexos. Esses *smart cards* terão o seu potencial expandido e melhor explorado à medida que a tecnologia dos semicondutores também evolua.

2.3.3 Cartões sem contato (*contactless card*)

Para a *Smart Card Alliance*, um *smart card* sem contato consiste é um *smart card* com um microcontrolador seguro embutido, ou inteligência equivalente, memória interna e

uma pequena antena e que se comunica com uma leitora através de um interface de rádio freqüência.

Neste tipo de *smart card* a transferência de energia e dados se dá sem qualquer contato físico com a leitora. Segundo Rankl e Effing (2003, pág. 8), os cartões sem contato podem ser tanto cartões de memória como cartões microprocessados. Entretanto, existem diferenças entre eles no que tange a distância de operação do terminal ou leitora. Os *smart cards* microprocessados devem trabalhar a uma distância de apenas alguns centímetros da leitora, enquanto os *smart cards* de memória podem ser operados a uma distância de até um metro do terminal. Ou seja, cartões de memória não precisam necessariamente estar na mão do usuário para que sejam utilizados. Desde que estejam próximos da leitora, podem estar dentro de bolsas ou carteiras.

Cartões sem contato, necessariamente, devem possuir uma antena integrada ao corpo do *chip*. Um dos métodos utilizados para a construção da antena consiste em embutir um rolo de fio fino de cobre no corpo do chip, formando assim uma espécie de bobina (RANKL, 2007, pág. 5).

Existem, ainda, os *smart cards* classificados como híbridos. Estes cartões são dotados de duas interfaces de contato, sendo uma de contato físico e outra sem.

A figura 2.4 ilustra a estrutura típica de um *smart card* microprocessado, com co-processador e duas interfaces, uma de contato e outra sem contato físico. As setas na figura indicam o fluxo básico de dados e energia dentro do *chip*.

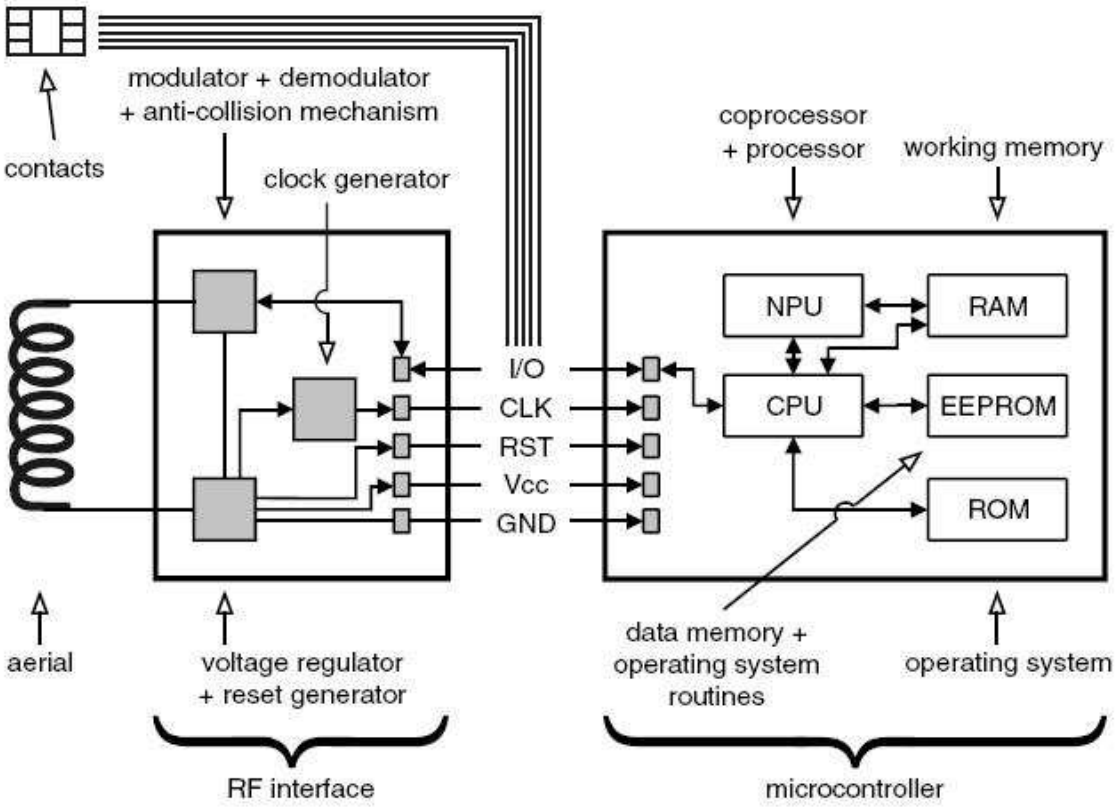

Figura 2.4 – Estrutura típica de um *smart card* microprocessado, com co-processador, uma interface de contato e outra sem contato físico.
Fonte: RANKL e EFFING (2003, pág. 20).

Dadas as suas caracteríticas, *smart cards* sem contato são tipicamente utilizados em aplicações que exijam autenticação rápida, como o transporte público, controle de acesso, identificação de bagagens, passaportes, praças de pedágio, estacionamentos, etc.

Devido ao escopo e objetivo do presente trabalho, as propriedades de transferência de dados e energia em cartões sem contato não serão abordados aqui. Para detalhes quanto às características deste tipo de *smart cards* recomenda-se a leitura da obra *Smart card Handbook*, de Rankl e Effing (2003, 1088 p.), título 3.6.

2.4 Formato de cartões

Os tipos mais comuns de cartões possuem uma característica em comum que é a espessura de 0,76 mm. Assim como ilustrado na figura 2.5, as demais dimensões dos cartões podem variar. Porém, essas variações são especificadas por normas internacionais ou por especificações publicadas pelos grandes emissores de cartões (RANKL, 2007, pág. 2). Cabe salientar que, no caso dos cartões de contato, essa padronização é essencial para que os cartões sejam compatíveis e possam encaixar e ser lidos por leitoras e terminais de diferentes fabricantes.

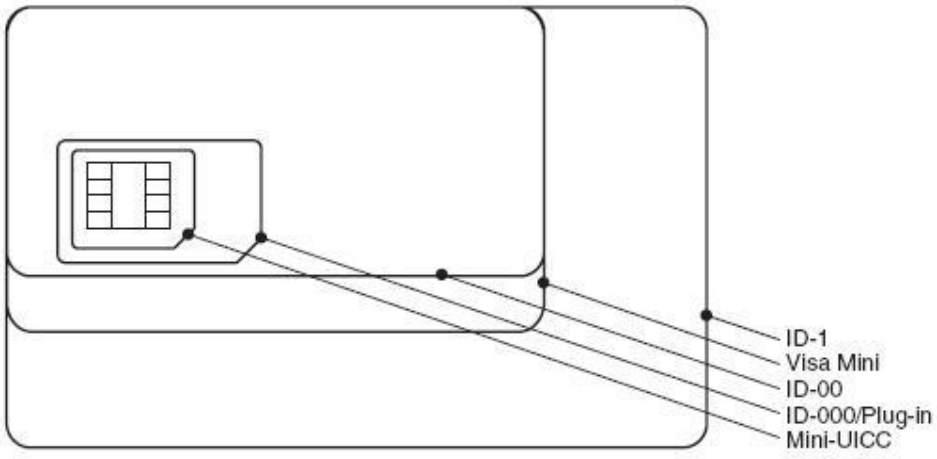

Figura 2.5 – Relação entre o tamanho dos tipos mais comuns de cartões.
Fonte: RANKL (2007, pág. 2).

Os cartões ID-1 são os mais conhecidos e utilizados, pois possuem as mesmas dimensões dos atuais cartões de crédito, cartões bancários e diversos outros cartões amplamente distribuídos entre a população. Os cartões Visa Mini são uma versão reduzida do padrão ID-1 e buscam oferecer cartões de pagamento com dimensões menores. Os cartões ID-000 são o padrão de cartões utilizados nos telefones móveis padrão GSM/UMTS e compreendem os SIM/USIM *cards*, enquanto os cartões Mini-UICC, também destinados ao setor de telecomunicações, são uma resposta a atual tendência de miniaturização dos aparelhos utilizados na telefonia móvel. A tabela 2.1 consolida os formatos mais comuns de cartões e suas dimensões.

Tabela 2.1 – Dimensões dos tipos de cartões mais comuns.

Formato do cartão	Largura (mm)	Altura (mm)	Uso
ID-1	85,6	54	Padrão conhecido
ID-00	66	33	Padronizado para telecomunicações, mas não utilizado
Visa Mini	65,6	40	Sistemas de pagamento
Plug-in, ID-000	25	15	Telecomunicações
Mini-UICC	15	12	Telecomunicações

Fonte: RANKL (2007, pág. 3).

2.5 Estrutura e componentes de um *smart card*

As características de um *smart card* são determinadas pelo seu microcontrolador. Conforme Rankl (2007, pág. 5), um microcontrolador consiste em um *chip* de silício com funções específicas e necessárias para seu uso, adaptadas conforme sua finalidade. As adaptações compreendem parâmetros elétricos e físicos como o consumo máximo de corrente elétrica, a faixa de *clock* permitida para o processador, além da faixa de temperatura admissível.

2.5.1 Módulo do *chip*

O módulo do *chip* é a proteção para o microcontrolador do *smart card*. O microcontrolador normalmente está localizado na parte traseira do módulo se ligado a superfície do mesmo através fios finíssimos. O módulo pode ter de seis a oito contatos físicos com a superfície exterior, apesar dos *chips* modernos necessitarem de apenas cinco contatos para o seu funcionamento. Os contatos adicionais estão reservados para aplicações futuras. Através destes contatos são transmitidos todos os sinais elétricos para o *chip*. Esta estrutura de contatos é padronizada pela norma ISO/IEC 7816-2.

A figura 2.6 ilustra os contatos físicos do módulo de um *chip* de *smart card*, com suas respectivas funcionalidades, a saber: Vcc – *supply voltage*, RST – *Reset signal*, CLK – *clock signal*, AUX1 – *auxiliary 1* (reservado para uso futuro), GND – *Ground*, SPU – *Standard or proprietary use*, I/O – *data input/output*, AUX2 – *Auxiliary 2* (reservado para uso futuro).

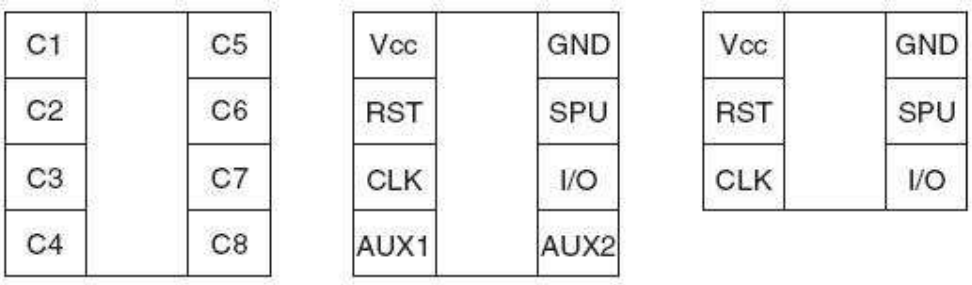

Figura 2.6 – Arquitetura de contatos do módulo do *chip*, conforme a ISO/IEC 7816-2.
Fonte: RANKL (2007, pág. 5).

2.5.2 Processador

As CPUs de *smart cards* são programadas para executar instruções do sistema operacional, localizado na memória ROM e podem ser auxiliadas por uma NPU para tarefas de cálculos numéricos, particularmente relacionados a criptografia. Guardadas as restrições, estes processadores combinam um alto poder de processamento com um baixo consumo de energia.

A maioria dos atuais *smart cards* são equipados com processadores de 8 bits, baseados em processadores Intel 8051[7]. O poder de processamento destes CPUs normalmente é suficiente para sistemas operacionais que não incluem um interpretador. No entanto, se o sistema operacional precisa prover suporte a um interpretador Java (JCVM), há uma preferência por processadores de 16 bits, normalmente, também baseados no Intel 8051. Ainda existem *smart cards* baseados em processadores de 32 bits, tais como os ARM 7 ou MIPS (RANKL, 2007, pág. 8).

Segundo Rankl (2007, pág. 8), um fator limitante para o uso de processadores de alta performance é a área do *chip*, pois existe uma relação parcialmente direta entre a área do *chip* e o seu preço. Tendo em vista que processadores de 32 bits ocupam uma área significativamente maior do que aqueles de 8 bits, muitas vezes é mais econômico investir em otimizar a velocidade do software do que na utilização de um *chip* que necessite de uma área maior.

[7] Popular processador de 8 bits lançados pela Intel em 1977. Atualmente possui diversos modelos clones sendo produzidos por empresas diversas à Intel. Por ser um microcontrolador CISC, oferece um conjunto de instruções muito vasto que permite executar desde um simples programa que faz piscar um LED até um programa de controle de acesso controlado por rede (Wikipedia, 2008).

2.5.3 Memórias

Além do processador, todo microcontrolador necessita de alguns tipos de memórias com diferentes características, dentre os quais uma memória não-volátil. Assim, a memória EEPROM (memória não volátil que pode ser substituída por memória *flash*) tem como finalidade armazenar os dados críticos que devem ser armazenados nos *smart cards* e que devem ser modificados durante uma operação. Já a memória RAM, da mesma forma que em um PC, funciona como memória de trabalho, armazenando dados voláteis durante as operações, enquanto a memória ROM armazena dados não-voláteis como instruções do sistema operacional.

Devido às restrições físicas do módulo do *chip*, todos os componentes estão limitados a uma área total de 25 mm^2. Como conseqüência disso, a capacidade das memórias é bastante reduzida se comparada com outros meios de armazenamentos conhecidos e encontrados em PCs modernos. Assim, a capacidade da memória ROM fica na faixa de 16 a 400 KB, a memória EEPROM entre 1 e 500 KB, enquanto a memória RAM varia sua capacidade entre 256 *bytes* e 16 KB (RANKL, 2007, pág. 7).

2.5.4 Hardware suplementar

Além de processador e memórias, os microcontroladores de *smart cards* podem incorporar diversos outros hardwares com o objetivo de aumentarem o leque de funcionalidades. A figura 2.7 ilustra as possibilidades de hardwares suplementares e a maneira como são ligados à CPU por meio de barramentos compartilhados.

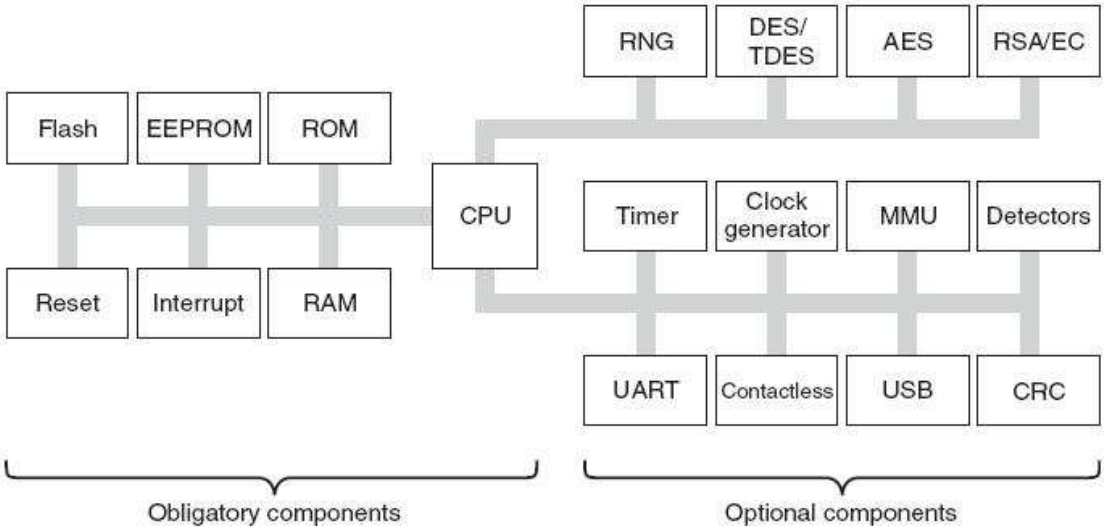

Figura 2.7 – Hardwares suplementares em um microcontrolador de *smart cards*.
Fonte: RANKL (2007, pág. 9).

Dentre os hardwares suplementares está o gerador de *clock*. Normalmente o sinal de *clock* necessário para a execução da CPU de um *smart card* é provido pelo terminal. Entretanto, devido às restrições de faixa de sinal *clock*, que deve variar entre 1 e 5 MHz, cada vez mais os microcontroladores estão incluindo um gerador de *clock* interno em sua arquitetura (RANKL, 2007, pág. 8).

Outro componente suplementar dos microcontroladores de *smart cards* é o UART (*Universal Asynchronous Receiver Transmitter*), responsável pela comunicação serial entre o *chip* e o terminal.

No entanto, segundo Rankl (2007, pág. 9), a maior parte dos hardwares suplementares está relacionada a funções criptográficas, uma vez que são as tarefas que mais consomem capacidade de processamento. Da mesma forma, a geração de números aleatórios, também, na maioria das vezes, é realizada utilizando um hardware suplementar. Algoritmos de criptografia simétrica como DES, 3DES e AES normalmente estão presentes em hardware suplementares. Já os hardwares especializados para computação de algoritmos assimétricos não estão presentes em todos os microcontroladores, uma vez que sua implementação incrementa o preço dos mesmos. Quando presentes, eles suportam algoritmos como RSA, DSA e ECC.

2.6 Segurança em s*mart cards*

Dentre todas as características e funcionalidades dos *smart cards*, as mais importantes são aquelas voltadas para as funções de segurança. Desde a sua concepção, os microcontroladores de *smart cards* foram especialmente desenvolvidos para suportarem ataques. Sua segurança inclui sistema de detecção de voltagem e/ou *clock* acima e abaixo da faixa especificada, além de sensores de luz e temperatura que permitem reconhecer ataques através destes meios e, então, fornecer uma reação adequada (RANKL, 2007, pág. 5).

Segundo Rankl (2007, pág. 5), ainda existem mecanismos de segurança relativamente mais complexos, como aqueles que compreendem o "embaralhamento" dos dados das memórias e dos barramentos entre o processador e a memória. Ainda, é possível que a chave de "embaralhamento" seja trocada periodicamente durante uma sessão individual. Os microcontroladores também podem defender-se de ataques pesados, como aqueles que consistem em medir o atual consumo de energia do *chip* a fim de realizar uma análise estatística e descobrir que dados foram processados pelo processador.

Percebe-se que todas estas características fazem dos *smart cards* módulos realmente seguros e adequados para o armazenamento e processamento de informações críticas.

2.7 Normas técnicas associadas à *smart cards*

Dada a diversidade de fabricantes de *smart cards* e a variedade de aplicações para os quais os mesmos podem ser utilizados, é indispensável que suas características sejam mundialmente padronizadas, de forma a garantir a sua integração e utilização em larga escala.

Desta forma, com o objetivo de suprir essas necessidades, surgiram padrões técnicos como a família de normas ISO/IEC 7816 e a especificação EMV.

2.7.1 ISO/IEC 7816

A família de normas técnicas ISO/IEC 7816 compreende um conjunto de padrões internacionalmente aceitos para *smart cards*. Segundo Ortiz (2003), o conjunto de normas ISO/IEC 7816, introduzida em 1987, define vários aspectos para *smart cards*, incluindo características físicas, contatos físicos, sinais elétricos, comandos, arquitetura de segurança, identificadores de aplicações e elementos de dados comuns.

Esta norma é dividida em diversas partes, cada qual abordando características específicas. A tabela 2.2 descreve partes desta norma e o foco das suas padronizações.

Tabela 2.2 – Norma ISO/IEC 7816 e suas divisões.

Norma ISO/IEC	Aplicação
7816-1	Características físicas dos cartões
7816-2	Dimensões e localização dos contatos
7816-3	Sinais eletrônicos e protocolos de transmissão
7816-4	Comandos para comunicação – protocolo APDU
7816-5	Sistema de numeração e identificadores de aplicação
7816-6	Dados para intercâmbio inter-indústria
7816-7	Comandos inter-indústria para SCQL
7816-8	Comandos para operações seguras
7816-9	Comandos para gerenciamento de cartões
7816-10	Sinais eletrônicos e ATR para cartões síncronos
7816-11	Verificação pessoal através de métodos biométricos
7816-12	Cartões com contato USB – Interfaces elétricas e procedimentos operacionais
7816-13	Comandos para gerenciamento de aplicações em ambientes multi-aplicações
7816-15	Aplicações de assinaturas criptográficas

Fonte: ISO/IEC 7816.

2.7.2 EMV 4.1

A EMV compreende um conjunto de normas internacionalmente aceitas criadas a partir da cooperação de três grandes operadoras de cartões de crédito da época, a saber: *Europay, Visa, Mastercard*. Conforme a EMVCo[8], suas especificações definem padrões para a interoperacionalidade de *smart cards* e terminais POS ou ATM para autenticação de transações financeiras. Com a aquisição da *Europay* pela *Mastercard* em 2002 e o ingresso da JCB (*Japan Card Bureau*) em 2004, atualmente a EMV é operada pelas empresas JCB *International, Mastercard Worldwide* e *Visa, Inc*.

A especificação EMV 4.1 consiste de quatro livros, os quais são relacionados na tabela 2.3 juntamente com uma breve descrição.

[8] http://www.emvco.com

Tabela 2.3 – Especificação EMV 4.1 e seus quatro livros

EMV 4.1	Objetivo
Book 1 – Application Independent Interface Requirement	Descreve os requisitos mínimos para cartões de circuitos integrados (ICC) e terminais para garantir o funcionamento e a interoperabilidade do conjunto independente da aplicação.
Book 2 – Security and Key Management	Descreve funcionalidades mínimas de segurança entre cartões de circuitos integrados e terminais, bem como requisitos e recomendações sobre comunicação *on-line*, gerenciamento de chaves criptográficas e sistemas de pagamentos.
Book 3 – Application Specification	Define procedimentos necessários a serem aplicados entre terminais e cartões de circuitos integrados para efetuar uma transação em um sistema de pagamento internacional.
Book 4 - Cardholder, Attendant, and Acquirer Interface Requirements	Define requisitos obrigatórios, recomendados e opcionais para terminais para suportar cartões de circuitos integrados que estejam em acordo com as definições dos livros 1, 2 e 3.

Fonte: adaptado de EMV*CO* – www.emvco.com, acessado em 01 de junho de 2008.

2.8 *Smart card* nas telecomunicações

Segundo Rankl e Effing (2003, pág. 724), uma das primeiras redes de telefonia móvel a utilizar *smart cards* na autenticação de seus assinantes foi a rede Alemã C-Netz, ainda no início da década de 1988.

A C-Netz é considerada a primeira geração da telefonia móvel - uma tecnologia 1G - e ainda utilizava transmissão de dados analógica na sua comunicação.

Atualmente os SIM *cards* são utilizados nas redes de telefonia GSM com a função básica de realizar a autenticação do cliente o *roaming*[9]. A evolução das redes GSM – as redes 3Gs, introduziram também a utilização do USIM *card*.

[9] *Roaming* é uma opção de serviço da rede GSM que permite receber e fazer ligações fora da área de cobertura da operadora contratada, utilizando a operadora local (pt.wikipedia.org/wiki/Roaming, acessado em 17/06/2008).

2.9 O sistema GSM (*Global System for Mobile*)

Muitas redes de telefonia móvel 1G surgiram na Europa na década de 1980. Porém a heterogeneidade existente entre os sistemas de diferentes países europeus impedia que os telefones celulares fossem utilizados fora das divisas da rede de determinado país. E foi com o propósito de resolver este problema que, depois de anos de pesquisa, surgiu a *Global System for Mobile Communications* (GSM).

O primeiro sistema GSM entrou em fase de testes em 1991 na Europa, passando a ser operado comercialmente em 1992. Esta tecnologia baseia-se na utilização de *smart cards* para a autenticação dos assinantes da rede de telefonia. Estes cartões são conhecidos como SIM *cards* (*Subscriber Identity Module*) quando na tecnologia 2G (GSM) e 2.5G (GPRS e EDGE) e USIM (*Universal Subscriber Identity Module*) quando na tecnologia 3G.

Em 1998, o *Subscriber Identity Module Expert Group* (SIMEG) iniciou os trabalhos sob uma especificação para os *smart cards* GSM (SIM *cards*). Este grupo era formado por representantes de fabricantes de *smart cards*, fabricantes de telefones móveis e operadoras de redes telecomunicações móveis. Trabalhando com o aval da ETSI (*European Telecommunication Standards Institute*) o SIMEG gerou a especificação para a interface entre os dispositivos móveis e os SIM *cards*, sob o nome de GSM 11.11. Em 1994, o SIMEG transformou-se no *Special Mobile Group 9* (SMG9), que desenvolveu e manteve as especificações para SIM *cards* até o ano 2000. Em 2000, o SMG9 foi dissolvido e suas responsabilidades dividas entre o ETSI *Project Smart card Plataform* (EP SCP), responsável por publicações genéricas para *smart cards* nas telecomunicações, e a 3GPP (*3rd Generation Partenership Project*), responsável por aplicações específicas de interface entre os dispositivos móveis e cartões SIM e USIM.

A evolução e o domínio das redes GSM são incontestáveis. Conforme anunciado pela GSM *Association*[10], através de publicação no seu site oficial[11], em 16 de abril de 2008 foi superada a marca de 3 bilhões de conexões à redes GSM (incluindo toda a família de tecnologias GSMs: GSM, GPRS, EDGE, W-CDMA, e HSPA) ao redor do mundo, distribuídos em redes de mais de 700 operadoras de telefonia, em 218 países. E este número está em constante crescimento, seguindo uma taxa de incremento de 15 novas conexões por

[10] A GSM Association (GSMA) é uma associação comercial que representa mais de 700 operadoras de telefonia móvel GSM em 218 países ao redor do mundo. Além disso, mais de 200 fabricantes apóiam as iniciativas da associação como parceiros chaves.
[11] http://www.gsmworld.com/news/press_2008/press08_31.shtml

segundo, resultando em cerca de 1,3 milhões ao dia. Segundo a mesma fonte, o Brasil contribui com 93 milhões de conexões à redes GSM.

2.10 O SIM *Card* (*Subscriber Identity Module Card*)

Conforme definido na GSM 02.17, "o SIM é uma entidade que contém a identidade do assinante. A função primária de um SIM é assegurar a autenticidade de uma estação móvel com a respectiva rede".

Segundo Rankl e Effing (2003, pág. 745), além das funções primárias de proteger a identidade do assinante, realizada através do PIN (*Personal Identification Number*), e autenticar o dispositivo móvel na rede GSM, o SIM possui outras funcionalidades, dentre as quais a de armazenar informações de forma segura, protegidas contra manipulação, e permitir a criação de serviços adicionais seguros através da rede de telecomunicações.

Inicialmente os SIM *cards* GSM foram projetados no formato ID-1, com dimensões semelhantes às de um cartão de crédito. Porém, a evolução e a conseqüente compactação dos telefones celulares fizeram com que os cartões no formato ID-000 se tornassem o padrão de *smart cards* para a telefonia móvel. As figuras 2.8 e 2.9 ilustram um SIM *card* no formato ID-1 com recorte no formato ID-000 que facilita a sua utilização em telefones celulares.

Figura 2.8 – SIM *card* GSM. Cartão no formato ID-1 com recorte no formato ID-000
Fonte: Schreiber, Souza, Garcia (2003).

Figura 2.9 – Cartão com recorte ID-000 para destacar o SIM *card*.
Fonte: http://www.m-99.co.uk/Mobile_Phone_SIM_*Card*s/Holder/holder.html

Conforme Martinez, Martinez, Montesinos e Skarmenta (2007), atualmente os SIM *cards* são cartões multi-aplicações, onde cada aplicação provê diferentes serviços que podem ser executados ao mesmo tempo.

Segundo a Sun Microsystems[12], quando equipado com tecnologia *Java Card*, os SIM *cards* também podem proporcionar serviços transacionais, como operações bancárias remotas. Centenas de milhões de SIM *cards* baseada na tecnologia *Java Card* já estão proporcionando a implementação de serviços inovadores em telefones celulares.

2.11 Especificações e padrões para redes GSM e SIM *Cards*

A especificação que apresenta os princípios de segurança para um módulo GSM é denominada GSM 02.17 (SIM *Functional Characteristics*) e, conforme Rankl e Effing (2003, pág. 738), contém uma descrição relativamente abstrata dos requisitos funcionais para SIM *cards*. A especificação mais importante, a GSM 11.11 (*Specification of the Subscriber Identity Module – Mobile Equipment* (SIM – ME) *interface*), é baseada nessa especificação.

As especificações para padrões elétricos, que complementam a GSM 11.11, estão descritas nos documentos GSM 11.12 (*Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface*) e GSM 11.18 (*Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface*).

Complementando as especificações anteriores que descrevem o funcionamento básico dos SIM *cards*, está a GSM 11.14 (*Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface*) que descreve a plataforma para o desenvolvimento de serviços suplementares seguros ao SIM. Ela faz referência ao SIM *Application Toolkit* (SAT). Esta especificação foi publicada em 1996 e, em princípio, oferecia às operadoras de redes de telefonia a possibilidade de carregarem as suas próprias aplicações dentro do *smart card*.

As especificações GSM 02.48 (*Specification of security mechanisms for the SIM application toolkit, stage 1*) e GSM 03.48 (*Specification of security mechanisms for the SIM application toolkit, stage 2*) padronizam dois importantes mecanismos de segurança para SIM *card*s. A primeira apresenta mecanismos para a transmissão segura de dados via OTA (*Over*

[12] <http://java.sun.com/javacard/overview.jsp>. Acesso em 12 de jun. 2008.

The Air), enquanto a segunda descreve os mecanismos para o gerenciamento remoto de arquivos e gerenciamento remoto de *applets* em SIM *card*s.

A base para todos os sistemas operacionais de *smart cards* que executam programas em um SIM estão descritas na especificação GSM 02.19. Ela contém uma lista de serviços básicos de uma API para execução de códigos de programas em SIM *cards*. Baseados nesta norma, está a GSM 03.19, que especifica a implementação detalhada da API *Java Card* para SIMs baseada na especificação *Java Card* 2.1. Esta especificação é o documento chave para a utilização de *Java Card* em dispositivos GSM. Ela ainda é complementada pela especificação GSM 11.13 que descreve o ambiente e aplicações de teste, além de procedimentos e exemplos para testes de aplicações *Java Card* em SIM *cards*. A tabela 2.4 apresenta um resumo das principais especificações GSM.

Tabela 2.4 – Principais especificações para rede SIM *cards* e serviços GSM.

Especificação	Título
GSM 02.09	*Security Aspects*
GSM 02.17	*SIM Functional Characteristics*
GSM 02.19	*Subscriber Identity Module Application Programming Interface (SIM API); Service description; Stage 1*
GSM 02.48	*Specification of Security Mechanisms for the SIM Application Toolkit, Stage 1*
GSM 03.19	*Digital Cellular Telecommunications System (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2*
GSM 03.48	*Specification of Security Mechanisms for the SIM Application Toolkit, Stage 2*
GSM 09.91	*Interworking aspects of the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface Between Phase 1 and Phase 2*
GSM 11.11	*Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface*
GSM 11.12	*Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface*
GSM 11.13	*Test Specification for SIM API for Java Card*
GSM 11.14	*Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface*
GSM 11.17	*Subscriber Identity Module (SIM) Conformance Test Specification*
GSM 11.18	*Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface*

Fonte: RANKL e EFFING (2003, pág. 739).

2.12 Arquitetura lógica de um SIM *card Java Card*

Do ponto de vista lógico, os SIM *cards* possuem sua estrutura de arquivos distribuída entre as memórias ROM e EEPROM, cuja ilustração é apresentada na figura 2.10. A memória ROM compreende a camada "física" do cartão e é o local onde residem o sistema operacional e as interfaces para entrada e saída. Logo acima está a máquina virtual *Java Card* (JCVM) – responsável pela interpretação dos *applets Java Card*, o *Card Manager* – responsável pelo gerenciamento do ciclo de vida de cada uma das aplicações, o SIM *Toolkit Security* – responsável por adicionar um cabeçalho de seguranças às mensagens SMS, além de diferentes APIs de desenvolvimento como *a JavaCard* e o *SIM Toolkit.* Nesta região da memória ROM ainda reside a aplicação GSM gravada pelo fabricante e que não pode ser removida (MARTINEZ et al., 2007, pág. 97).

Por sua vez, a memória EEPROM abriga as diversas aplicações de um cartão multi-aplicação. Esta zona de memória pode ser modificada durante o ciclo de vida do SIM *card* através do *card manager*. Conforme ilustrado na figura 2.10, dentre os arquivos e aplicações abrigados na EEPROM estão os arquivos GSM que contêm as chaves GSM, o catálogo de endereços e mensagens SMS, *applets* USAT que são aplicações desenvolvidas com a tecnologia SIM *Application Toolkit*, aplicações WIM voltadas para a execução de operações criptográficas, além de outros *applets Java Card* que podem ser carregados (MARTINEZ et al., 2007, pág. 97).

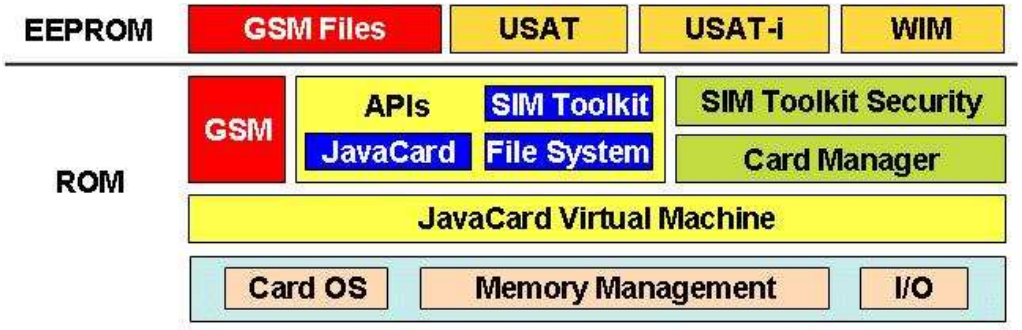

Figura 2.10 – Arquitetura lógica de um SIM *card*.
Fonte: Martinez et al. (2007, pág. 97).

2.12.1 SAT – USAT (*SIM Application Toolkit – Universal SIM Application Toolkit*)

O SIM *Application Toolkit* (SAT) define um conjunto completo de comandos e eventos para a interação entre um dispositivo 2G e um SIM *card* 2G. Essa interface de comunicação facilita o desenvolvimento de novos serviços baseados em SIM *cards*, independentes dos fabricantes dos dispositivos móveis ou dos cartões. Dentre as capacidades de uma aplicação baseada em SIM *card* está a de mostrar diferentes itens de menus nos dispositivos móveis ou ainda poder iniciar um processo de comunicação através do dispositivo, estabelecendo uma chamada telefônica ou enviando uma mensagem SMS (MARTINEZ et al., 2007, pág. 98).

A evolução do SAT é o USAT (*Universal* SIM *Toolkit Application*), utilizado pelo padrão de 3ª geração estabelecido como evolução para as redes GSM. O USAT possui os mesmos princípios e conceitos da versão anterior, porém, com algumas melhorias, como a possibilidade de abertura de uma conexão HTTP com um outro dispositivo com endereço IP a partir de comandos pró-ativos.

Segundo Martinez et al. (2007, pág. 98), normalmente uma aplicação SIM Toolkit é um *applet Java Card*.

2.12.2 WIM (*Wireless Identity Module*)

Wireless Identity Module é uma especificação de segurança que define como armazenar e gerenciar credenciais criptográficas em um SIM *card*. Ela define a forma segura para executar processos de assinatura eletrônica em um SIM *card tamper-resistant*[13]. O WIM é definido como uma aplicação independente dentro do cartão, assim como uma aplicação GSM ou os *applets* SAT. Desta forma, ele pode ser utilizado para realizar operações criptográficas de diferentes protocolos de dispositivos móveis, como o TLS (*Transport Layer Security*) e o S/MIME[14] (*Secure / Multipurpose Internet Mail Extensions*), e para diferentes aplicações do SIM *card* multi-aplicação (Martinez, Martinez, Montesinos e Skarmenta, 2007, pág. 98).

[13] Um módulo *tamper-resistant* é um dispositivo que possui proteções de hardware contra engenharia reversa e extração das informações nele contidas.
[14] S/MIME é um padrão de segurança para assinatura encriptação com modelo de chaves públicas de e-mails, fornecendo recursos de autenticação, integridade de mensagens e não repúdio.

Para Rankl e Effing (2003, pág. 802), a principal aplicação do WIM é a de prestar funções criptográficas para garantir o transporte seguro de informações em transmissões WAP. O WIM é baseado na especificação PKCS#15 (RSA) para aplicações de assinatura digital em *smart cards* e é compatível com a norma ISO/IEC 7816-15. Além do contexto WAP, o WIM pode ser utilizado no ambiente SIM *Application Toolkit* com o propósito de disponibilizar aplicações de assinatura em cartões de telecomunicações. Por este motivo as aplicações WIM se tornaram um padrão em muitos SIM e USIM *cards*.

Um SIM *card* que possui uma aplicação WIM pode ser considerado um SSCD (*Secure Signature Creation Device*), pois os pares de chaves criptográficas são gerados dentro do cartão, o processo de assinatura é executado pela aplicação WIM e as chaves privadas nunca saem do cartão. Por essas razões, uma assinatura eletrônica obtida através de uma aplicação WIM pode ser entendida como uma "assinatura qualificada". Assim, esta tecnologia pode ser considerada como uma das bases para o desenvolvimento de aplicações de assinatura eletrônica no que diz respeito à geração e gerenciamento de chaves. Atualmente, os fabricantes de SIM *cards* estão incluindo aplicações WIM em seus cartões como um padrão de assinatura eletrônica (Martinez et al., 2007, pág. 99).

2.12.3 USAT – i (*Universal SIM Application Toolkit – Interpreter*)

O *Universal* SIM *Application Toolkit – Interpreter* é uma nova aplicação SAT para SIM *cards* 3G. A aplicação USAT-*i* é um interpretador de *byte-codes* que trabalha como um pequeno *browser* dentro de um SIM *card*. O *browser* USAT-*i* recebe *byte-codes* através de mensagens SMS, os processa e, então, apaga-os. Esta tecnologia é suportada através de uma infra-estrutura de rede composta por um servidor de aplicação, um *gateway* e o dispositivo móvel. O servidor de aplicação hospeda páginas escritas em WML (*Wireless Mark-up Language*) que podem conter *tags* de definição de interface com o usuário ou podem chamar comandos SAT. O *gateway* solicita uma página WML ao servidor e a transforma em *byte-codes* que, normalmente, são enviados ao USAT-*i* através de mensagens SMS. Os *byte-codes* são otimizados para serem transmitidos por uma pequena largura de banda estão prontos para serem interpretados pelo USAT-*i*. Atualmente, as mensagens SMS são o meio mais utilizado para a distribuição de *byte-codes*. Entretanto, as redes 3G oferecem protocolos como o TCP/IP ou HTTP para estabelecer a comunicação entre o *gateway* e o SIM *cards*.

A 3GPP[15] (*3rd Generation Partenership Project*) define que os browser dos USAT-*i* podem ser incrementados pelos fabricantes através de *plug-ins*. Um *plug-in* USAT-*i* é um *applet Java Card* que implementa uma interface compartilhável, instalada na fase de fabricação do *chip*, que permite ao browser USAT-*i* invocar seus comandos. Normalmente, os fabricantes que fornecem *plugins* USAT-*i* implementam *plug-ins* de PKI. Estes *plugins* especiais são aptos para trabalharem com a aplicação WIM e realizar processos criptográficos. O principal *plugin*, chamado *Assimetric Decryption* (AD), é utilizado para decriptações assimétricas, enquanto o *plugin* PKCS#7 para assinatura assimétrica. Já o *plugin Fingerpoint* para assinaturas assimétricas de *arrays* de *bytes* no formato PKCS#1. Estes dois últimos *plugins* fornecem o mecanismo para realizar assinaturas eletrônicas através de um *browser* USAT-*i* em um SIM *card*. A figura 2.11 ilustra a troca de informações entre o USAT-*i* e o servidor de aplicação.

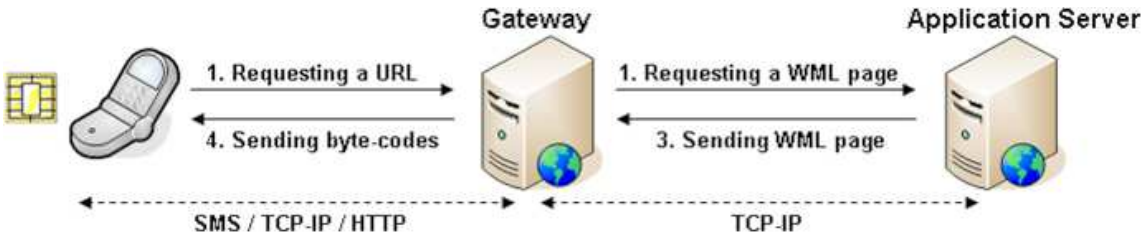

Figura 2.11 – Infra-estrutura associada ao funcionamento de um USAT-*i*.
Fonte: Martinez et al. (2007, pág. 98).

Assim, com a utilização de *browsers* USAT-*i* a comunicação com o SIM *card* é realizada sem a necessidade de qualquer aplicativo no dispositivo móvel que o hospeda.

2.13 OTA - *Over the Air*

Muitas vezes, depois que um cartão SIM foi emitido pelo fabricante, é necessário estabelecer um contato direto com o SIM *card* para gerenciamento das aplicações existentes e adição de novos serviços. Para suprir essa necessidade foram criados alguns mecanismos, os quais são especificados pela GSM 03.48. Estes mecanismos permitem uma comunicação

[15] A 3GPP é um acordo de colaboração criado em 1998 que visa reunir padrões e especificações de organismos da área de telecomunicações. Seu o objetivo é produzir especificações e relatórios técnicos para a terceira geração dos sistemas de telefonia móvel (3G), bem como para a tecnologia GSM, GPRS e EDGE.

segura entre os sistemas de apoio das operadoras de telefonia e o SIM *card* através de uma interface aérea e são chamados de comunicação OTA (*Over The Air*). Um exemplo de funcionalidade para o mecanismo OTA é o *download* de códigos de programas executáveis sob a forma de *applets* para complementar ou adicionar valor às aplicações de SIM *cards Java Card*. Um dos meios mais utilizados pelo OTA são as mensagens SMS disponíveis na rede GSM. As mensagens SMS são, então, utilizadas como recipientes para a troca de dados entre os sistemas de apoio e o SIM *card*.

O mecanismo OTA oferece serviços que garantem a segurança das informações transmitidas. O nível de segurança mais simples consiste em usar um *checksun* (CRC) para proteger os dados contra erros de transmissão. Se necessário também podem ser utilizados algoritmos criptográficos, como DES ou 3-DES, ou ainda assinaturas digitais.

O princípio operacional do uso de SMS para envio de comandos para um determinado SIM *card* segue os passos descritos a seguir. Inicialmente o sistema de apoio gera um SMS para o cartão em questão e carrega um comando nesta mensagem, se necessário utilizando mecanismos criptográficos. Tão logo o SIM em questão conecte-se a rede, a mensagem é transmitida e recepcionada pelo mesmo. Com base na codificação da mensagem, conforme especificado na GSM 03.40, o dispositivo móvel reconhece que a mensagem contém dados específicos destinados ao SIM e utiliza o comando ENVELOPE do SIM *Application Toolkit* para enviá-la ao SIM. A mensagem não é logo armazenada no sistema de arquivos do SIM, pois o SIM armazena os dados recebidos através do comando ENVELOPE em um *buffer* separado. Se o comando faz parte de uma seqüência de mensagens o SIM ainda deve restaurar a seqüência correta das mensagens, visto que elas podem chegar em ordem diferente daquela em que foram enviados. Em seguida o SIM interpreta a mensagem recebida, extrais o(s) comando(s) recebido(s) e o(s) processa. Como resposta o SIM pode, facultativamente, enviar uma mensagem SMS de resposta.

3 A PLATAFORMA *JAVA CARD*

A seguir são apresentados os principais conceitos a cerca da plataforma Java Card e que servem de embasamento teórico para o desenvolvimento do protótipo alvo deste trabalho. São abordados tópicos que passam pela visão geral da plataforma *Java Card*, segurança em *Java Card*, elementos de uma aplicação *Java Card*, a JCVM (*Java Card Virtual Machine*), o JCRE (*Java Card Runtime Environment*), o protocolo APDU e a API JSR 177 – SATSA.

3.1 *Java Card* – Visão Geral

O surgimento da tecnologia *Java Card*, apresentada pela Sun Microsystems em 1988, trouxe um importante avanço no desenvolvimento de aplicações para *smart cards*, tornando as aplicações independentes do hardware ou do fabricante.

Segundo a Sun Microsystems[16], a tecnologia *Java Card* permite que *smart cards* e outros dispositivos com memória muito limitada executem pequenas aplicações chamadas *applets* ou, mais especificamente, *card applet* ou *cardlet* para distinguir dos *applets* executados pelos *browsers*. Ela fornece aos fabricantes de *smart cards* uma plataforma segura e interoperável que permite armazenar e executar múltiplas aplicações em um único dispositivo.

No entanto, devido às limitações destes dispositivos, não existiriam recursos de hardware para suportar todas as propriedades da linguagem Java. Assim, a plataforma *Java Card* suporta apenas um subconjunto personalizado de características da linguagem Java. Este subgrupo apresenta os recursos de software essenciais para o desenvolvimento de aplicações para dispositivos limitados e preserva as capacidades de orientação a objeto da linguagem Java (SUN MICROSYSTEMS, 2006, 1.1).

[16] http://java.sun.com/javacard/overview.jsp. Acesso em 16 de jun. 2008.

A base para a tecnologia *Java Card* está na JCVM (*Java Card Virtual Machine*). A máquina virtual *Java Card* traz a portabilidade da linguagem Java para o mundo dos *smart cards*, permitindo que a premissa “escreva uma vez, execute em qualquer lugar” seja também aplicada a esses dispositivos. Da mesma forma que JVM define os arquivos *.class* como compatibilidade binária da plataforma Java, a especificação JCVM define os arquivos no formato *.CAP* como padrão para a compatibilidade binária entre dispositivos que implementam uma JCVM.

A plataforma *Java Card* também possui recursos que permitem a criação de um ambiente seguro para a execução das aplicações, além de permitirem a possibilidade de instalar e executar diferentes *card applets* em um mesmo cartão (MARTINEZ et al., 2007, pág. 98).

Conforme Khan (2005), uma aplicação *Java Card* também pode funcionar como um módulo para autenticação em uma aplicação bancária J2ME. As aplicações de *mobile bank* permitem que correntistas acessem as contas bancárias usando seus telefones celulares. Assim a aplicação *Java Card* contém a autenticação lógica, através da encriptação dos dados do cliente e assinatura das operações, enquanto o dispositivo J2ME contém um MIDlet que irá apresentar uma interface amigável para acesso as transações bancárias.

Para Gemalto ([S.d], pág. 17), a utilização da linguagem Java para aplicações em *smart card* traz, entre outros, os seguintes benefícios:

a) Múltiplas aplicações em um mesmo cartão: a arquitetura do cartão e as características de segurança da linguagem Java tornam possível que múltiplas aplicações residam de forma segura no cartão. O número de aplicações residentes no cartão somente é limitado pela quantidade de memória do mesmo.

b) Orientação a objetos: programadores têm os benefícios da reutilização de códigos, padrões de projeto, além de uma estrutura superior.

c) Independência de plataforma – programas *Java Card* são portáveis em diferentes arquiteturas de chips, gerando um menor custo desenvolvimento e manutenção.

d) Atualizações dinâmicas – desenvolvimento e implementações de *applets* de forma gradativa acrescentando funcionalidades à medida que as mesmas são desenvolvidas. Ainda, *applets* podem ser incluídos ou excluídos de um cartão em qualquer parte do seu ciclo de vida.

3.2 Segurança em *Java Card*

Para a Gemalto ([S.d], pág. 17), a integridade e segurança da linguagem Java são amplamente conhecidas. A segurança no gerenciamento de aplicações desenvolvido para os *smart cards* é implementado pela JCVM. Segundo a Gemalto ([S.d], pág. 17), são disponibilizadas as seguintes funções de segurança e integridade:

a) A linguagem Java Card implementa um verificador de classes do arquivo Java, executado antes do código ser carregado no cartão.

b) O JCRE reforça a segurança através de firewalls que isolam os applets e impedem que objetos criados por um applet sejam acessados indevidamente por outros.

c) O compilador Java Card realiza um rigoroso controle de erros quando o programa é compilado. Todas as referências a métodos e variáveis são verificadas para garantir que os objetos são do mesmo tipo e que o programa não acesse variáveis não inicializadas.

d) Programas maliciosos não podem direcionar ponteiros para a memória porque não existem ponteiros que possam ser acessados pelos programadores ou usuários.

e) A tecnologia Java Card acessa as variáveis somente através de referências a elas a partir de uma pilha Java. Programas maliciosos são impedidos de examinar o heap da memória porque os valores das variáveis locais não ficam disponíveis depois de serem invocados por métodos. Um método não pode acessar recursos que não devem.

3.3 Isolamento de *applets* e compartilhamento de objetos

A plataforma *Java Card* é um ambiente multi-aplicação seguro, onde diferentes *applets* de diferentes desenvolvedores podem residir de forma isolada e segura (ORTIZ, 2003a).

Na plataforma *Java Card*, os *applets* são completamente isolados uns dos outros. Conforme ilustra a figura 3.1, *applets* localizados no mesmo *package* compartilham um mesmo ambiente de segurança. Qualquer possível interferência é controlada pelo *security manager* da *Java Card Virtual Machine*, enquanto o compartilhamento de objetos é

gerenciado pelo *Java Card Runtime Environment* (JCRE) (RANKL e EFFING, 2003, pág. 318).

Conforme a Sun Microsystems (2006, pág. 6-1), qualquer implementação do JCRE deve suportar o isolamento de contextos para um *applet*. Isso significa que um *applet* não pode acessar os campos ou objetos de um *applet* em outro contexto, a menos que esse outro *applet* fornece explicitamente uma interface para acesso.

Segundo Ortiz (2003a), durante a sua execução, cada *applet* é assinado para um contexto de execução que controla o acesso aos objetos assinados para este contexto. A fronteira criada entre um ambiente de execução e outro é chamado de *applet firewall*. A figura 3.1 ilustra o contexto de execução, compartilhamento de objetos e o *applet firewall*.

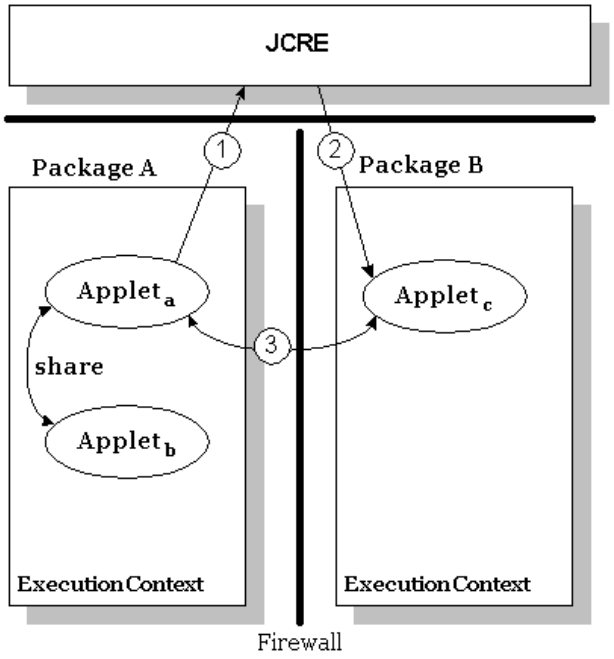

Figura 3.1 – Contextos de execução, compartilhamento de objetos e o *applet firewall*.
Fonte: ORTIZ (2003a).

Para maiores detalhes sobre a forma como a JCRE implementa um *applet firewall* e o compartilhamento de objetos em diferentes contextos recomenda-se a leitura da *Java Card Runtime Environment Specification, version 2.2.2*, publicada pela Sun Microsystems.

3.4 Elementos de uma aplicação *Java Card*

Segundo Ortiz (2003a), uma aplicação *Java Card* completa é formada pelos seguintes elementos: aplicações e sistemas *back-end*, uma aplicação *host* (*off-card*), um dispositivo para a interface entre a aplicação *host* e o cartão (leitora *smart card*) e o *applet Java Card*, além de credenciais de usuário. A figura 3.2 ilustra os elementos de uma aplicação *Java Card*.

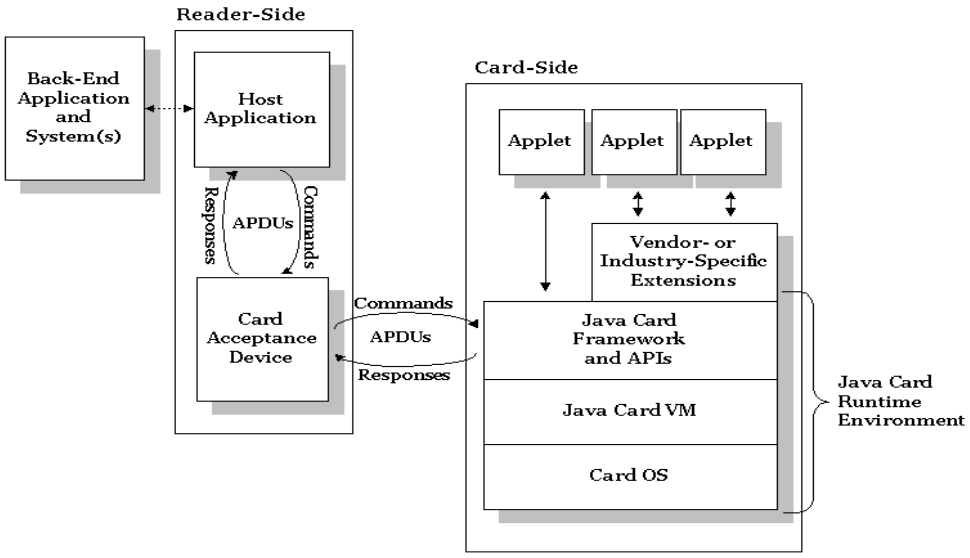

Figura 3.2 – Elementos de uma aplicação Java Card.
Fonte: ORTIZ (2003a).

3.4.1 Aplicação *back-end*

As aplicações *back-end* são aquelas que provêem serviços de suporte como informações de segurança e autenticação em sistemas de pagamento eletrônicos, na maioria das vezes interagindo com servidores de banco de dados.

3.4.2 Aplicações *host*

As aplicações *host* provêem a comunicação entre os usuários e os *applets Java Card*. Normalmente residem em PCs, terminais de pagamento eletrônico (POS) ou dispositivos móveis, como aparelhos celulares.

A plataforma J2ME permite o desenvolvimento de aplicações *host* através da utilização das APIs *Securing and Trust Service* (SATSA), possibilitando, assim, que aplicações desta plataforma acessem e interajam com *applets Java Card*. A plataforma J2ME aliada a API SATSA permitiu que aplicações residentes em SIM *cards* fossem utilizadas através de interfaces mais amigáveis, contribuindo para a expansão dos serviços de *mobile commerce* e *mobile banking*.

3.4.3 Aplicações *host* – leitoras *smart cards*

Um *Card Acceptance Device* (CAD), ou leitora *smart cards*, é um dispositivo de interface que se situa entre a aplicação *host* e o *smart card*. Ele provê a energia necessária para o funcionamento do cartão, bem como realiza a comunicação com o cartão através de contatos elétricos ou rádio freqüência (ORTIZ, 2003a).

Um dispositivo CAD pode ser uma leitora conectada a um PC através das portas USB ou serial, uma leitora integrada a um terminal de pagamento eletrônico ou, ainda, ser um componente em dispositivos móveis, como os aparelhos GSM, para leitura de SIM *cards*.

A função básica de um CAD é transmitir comandos APDU da aplicação *host* para o *applet* do cartão e transmitir as respostas do cartão para a aplicação *host*.

Alguns CAD possuem teclados numéricos para a digitação do PIN ou, ainda, interfaces para a leitura de informações biométricas, como impressões digitais.

3.4.4 *Applets* Java Card

A plataforma *Java Card* é um ambiente multi-aplicação, onde podem residir diferentes *applets* concomitantemente, conforme ilustram as figuras 3.1 e 3.2.

Todo *applet Java Card* estende a classe *Applet* e deve, obrigatoriamente, implementar os métodos *install()* e *process()*, pois os mesmos são invocados pelo JCRE, respectivamente, quando um *applet* é instalado e toda vez que é recebido um comando APDU para um *applet* (ORTIZ, 2003a).

Os *applets Java Card* são instanciados quando carregados no cartão e se mantém vivos mesmo após a energia ser desligada. Um *card applet* atua de forma passiva, semelhante a um servidor. Após o fornecimento de energia ao cartão pelo CAD, cada *applet* continua inativo até que seja selecionado, quando então pode ser inicializado. Um *applet* somente torna-se ativo quando recebe um comando APDU enviado pela aplicação *host* (ORTIZ, 2003a).

3.5 JCRE – *Java Card Runtime Environment*

A tecnologia *Java Card* define um *Java Card Runtime Environment* (JCRE) que contém um ambiente completo para suportar a execução dos *applets Java Card*. O JCRE contém uma *Java Card Virtual Machine*, a *Java Card Application Programming Interface* (API), que provê classes e métodos para o desenvolvimento de *card applets*, além de suporte a serviços. A figura 3.3 contextualiza o JCRE.

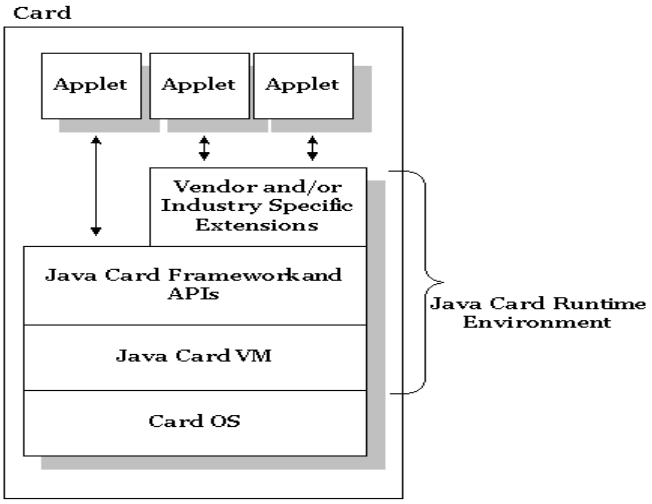

Figura 3.3 – Arquitetura de um *Java Card* e seu *Runtime Environment*.
Fonte: ORTIZ (2003a).

3.6 JCVM – *Java Card Virtual Machine*

A máquina virtual *Java Card* (JCVM) é uma versão da máquina virtual Java (JVM) adaptada para *smart cards*. Ela controla o acesso a todos os recursos do *smart card*, como memória e portas de entrada e saída, além de permitir que aplicações sejam carregadas de forma segura no cartão mesmo após a sua fabricação.

A JCVM executa um subconjunto de *byte codes* Java em *smart cards*, provendo, basicamente, acesso externo aos diversos recursos do cartão proporcionados pelas aplicações nele instaladas.

3.7 A API *Java Card*

As classes da API *Java Card* definem meios para desenvolver aplicações e prover serviços de sistema para essas aplicações. Essas classes definem convenções para que *applets* acessem a JCRE e funções nativas, como recursos do sistema operacional, acesso a memória e operações de entrada e saída. A API *Java Card* contém os seguintes pacotes, entre outros:

a) ***javacard.framework*** – define interfaces, classes e exceções que compõe o núcleo do *Java Card* Framework. Este package define importantes conceitos como o *Personal Identification Number* (PIN), o *Application Protocol Data Unit* (APDU), o *Java Card applet* (*Applet*), o *Java Card System* (*JCSystem*) e classes utilitárias. Também define várias constantes conforme especifica a norma ISO/IEC 7816, além de exceções para código *Java Card*.

b) ***java.io*** – define uma classe de exceção, a *IOException*. Nenhuma das outras classes tradicionais estão incluídas.

c) ***java.lang*** - define diversos tipos de exceções. Ela é importada automaticamente pelo compilador.

d) ***javacard.security*** – define classes e interfaces para funções de segurança em *Java Card*s. A especificação *Java Card* define uma API de segurança robusta que inclui vários tipos de algoritmos de chaves públicas e privadas, métodos para verificação de CRCs (*Cyclic Redundancy Check*) e assinaturas.

e) ***javacardx.crypto*** – define a interface *KeyEncriptation* e a classe *Cypher*. Ela usa o *KeyEncriptation* para decriptar uma chave recebida para ser usada pelos algoritmos de encriptação. *Cypher* é uma classe abstrata que todas os funções de cifragem devem implementar.

3.8 3GPP TS 43.019 API

Segundo a Gemalto ([S.d], pág. 19), as classes da API 3GPP TS 43.019 são uma extensão das classes da API *Java Card*. Elas permitem o desenvolvimento e o carregamento de *applets* que acessem as funções e dados em SIM *cards*. Possui os seguintes pacotes:

a) ***sim.access*** - define meios para que os *applets* acessem dados e arquivos da aplicações GSM definida na especificação 3GPP TS 51.011.

b) ***sim.toolkit*** – define meios para os *applets* manipulares informações e enviar comandos pró-ativos definidos na especificação 3GPP TS 51.014.

3.9 Comunicação com o *smart card*

A comunicação com um *smart card* sempre é iniciada pelo CAD, utilizando o modelo de comunicação *half-duplex*. O cartão sempre responde a comandos enviados pelo CAD, de maneira que o mesmo nunca enviará um comando de resposta ou dados sem antes receber um estímulo externo. Este modelo reproduz um relacionamento mestre-escravo entre o CAD (mestre) e o *smart card* (escravo). Os comandos pró-ativos enviados por um SIM *card* também são baseados nesse modelo de comunicação mestre-escravo (RANKL e EFFING, 2003, pág. 371).

Depois de inserido em um CAD, inicialmente os contatos do *smart card* são conectados mecanicamente ao terminal. Os cinco contatos são, então, carregados eletricamente seguindo a seqüência correta. Em seguida, o cartão automaticamente é “ligado”, realiza o seu *reset* e envia uma mensagem ATR (*Answer To Reset*) para o terminal. O terminal avalia o ATR que contém diversos parâmetros relacionados com o cartão e o protocolo de transmissão de dados e em seguida envia o primeiro comando. O cartão processa o comando e envia uma resposta ao terminal. Este ciclo de interação de comando e resposta continua até que o cartão seja desativado (RANKL e EFFING, 2003, pág. 371).

É possível utilizar dois modelos de protocolo de camada de aplicação para a comunicação entre a aplicação *host* e o *applet Java Card*. O primeiro baseia-se no modelo fundamental de transmissão de dados (*message-passing*) e o segundo é baseado no *Java Card Remote Method Invocation* (JCRMI), um modelo de objeto distribuído, subconjunto do J2SE RMI (ORTIZ, 2003a).

Dado o foco deste trabalho, aqui somente será abordado o primeiro modelo. Para maiores detalhes recomenda-se a leitura do livro *Smart card Handbook* de Rankl e Effing (2003).

3.10 Protocolo APDU

O modelo *message-passing*, é a base para da comunicação da plataforma *Java Card*. No centro deste modelo está o *Application Protocol Data Unit* (APDU), que nada mais é que um pacote de dados trocado entre o CAD e o *smart card*. O *smart card* recebe os comandos APDUs (C-APDU) do CAD e os encaminha para os *applets* apropriados. O *applet* processa o comando e envia uma resposta APDU (R-APDU) para o CAD (ORTIZ, 2003a). A figura 3.4 ilustra a comunicação entre a aplicação *host* e o *applet Java Card*.

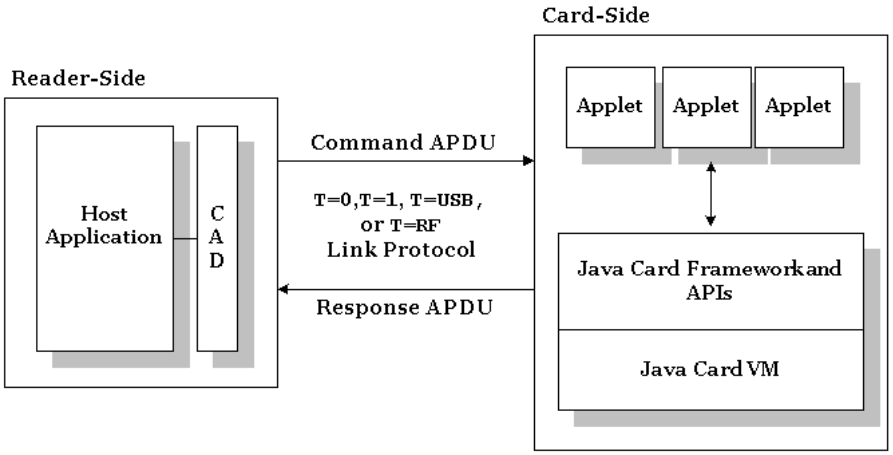

Figura 3.4 – Modelo de comunicação entre a aplicação *host* e o *applet Java Card*.
Fonte: ORTIZ (2003a).

O APDU é uma unidade de dados para a camada de aplicação. Ele é o equivalente a camada 7 do modelo OSI. No entanto, nos *smart cards* ele está localizado logo acima do protocolo de transmissão. Os protocolos APDUs *compliance* à norma ISO/IEC 7816-4 são independentes dos protocolos da camada de transmissão, assim, o conteúdo de um comando APDU não deve ser alterado em função do protocolo de transmissão.

A comunicação entre a aplicação *host* e o *applet* normalmente é baseada nos protocolos T =0 (*byte oriented*) ou T = 1 (*block oriented*). Entretanto, protocolos como o T = USB ou T = RF também podem ser utilizados.

3.10.1 Estrutura de um comando APDU

Um comando APDU é composto por um cabeçalho (*header*) e um corpo (*body*). O corpo pode ter um tamanho variável ou pode simplesmente não existir se não houver dados para serem associados aos campos do corpo (RANKL e EFFING, 2003, pág. 422).

A estrutura de um comando APDU é controlada pelo valor do seu primeiro *byte* e normalmente possuem o formato conforme ilustrado na figura 3.5.

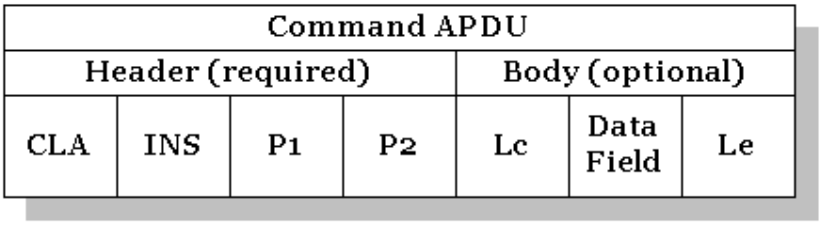

Figura 3.5 – Estrutura básica de um comando APDU.
Fonte: ORTIZ (2003a).

Conforme Ortiz (2003a), os campos ilustrados na figura 3.5 possuem as seguintes funções:

a) **CLA (1 *byte*)** – este campo é obrigatório e é utilizado para identificar aplicações e suas classes de comandos específicos. Os valores válidos para o campo CLA são especificados pela norma ISO/IEC 7816-4.

b) **INS (1 *byte*)** – este campo é obrigatório e identifica uma instrução específica para a classe de instruções identificada no campo CLA. Especifica as instruções básicas para acesso a dados no cartão quando o seu sistema de arquivos está estruturado conforme definem as normas ISO/IEC 7816.

c) **P1 (1 *byte*)** – este campo é obrigatório e define instruções para o parâmetro 1. Ele pode ser utilizado para qualificar o campo INS ou para entrada de dados.

d) **P2 (1 *byte*)** – este campo é obrigatório e define instruções para o parâmetro 2. Ele pode ser utilizado para qualificar o campo INS ou para entrada de dados.

e) **Lc (1 *byte*)** – este campo é opcional e indica o número de *bytes* contidos no campo data field.

f) ***Data Field*** (variável, indicado pelo campo Lc) – este campo é opcional e contém informações associadas ao comando solicitado.

g) **Le (1 *byte*)** – este campo é opcional e indica o número máximo de *bytes* esperados no campo *data field* do APDU de resposta.

Para exemplificar a forma de utilização de um comando APDU, a tabela 3.1 apresenta um exemplo de comando APDU utilizado para realizar a verificação do PIN.

Tabela 3.1 – Comando APDU – *Verify* PIN

Name	CLA	INS	P1	P2	Lc	Data Field	Lê (tamanho da resposta)
Verify PIN	0x80	0x30	0	0	Tamanho do PIN	Valor do PIN	N/A

Fonte: Adaptado de ORTIZ (2003b).

Dependendo da existência de dados no *data field*, ou se uma resposta é requerida pelo comando solicitante, existem quatro possíveis estruturas para um comando APDU. O desenvolvedor deve preocupar-se com estas variações somente se estiver usando o protocolo de transmissão T = 0 (ORTIZ, 2003). A figura 3.6 ilustra as quatro possibilidades de estrutura de comando APDU.

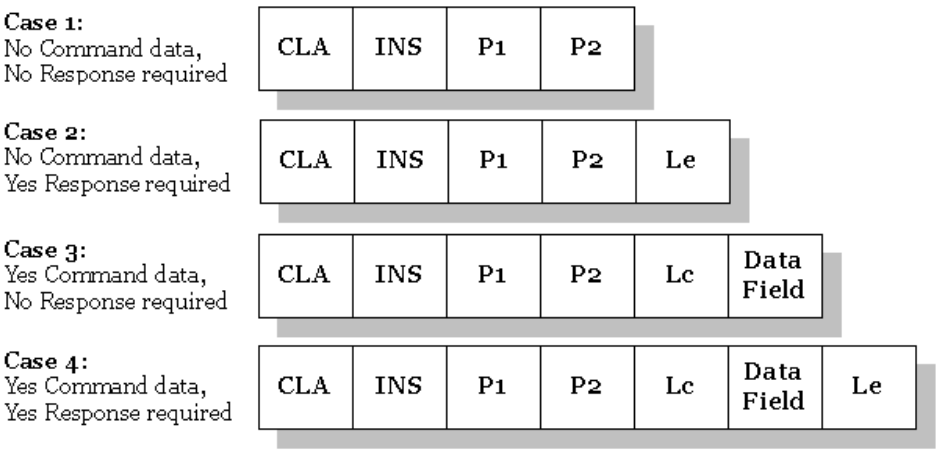

Figura 3.6 – Possibilidades de estruturas de um comando APDU.
Fonte: ORTIZ (2003a).

3.10.2 Estrutura de uma resposta APDU

Assim como o comando APDU, um APDU de resposta possui um corpo opcional e um *trailler* obrigatório. O *trailler* compreende os campos SW1 e SW2, enquanto corpo é composto pelo *data field*, cujo tamanho é especificado pelo *byte Le* do comando APDU precedente (RANKL e EFFING, 2003, pág. 424), conforme ilustra a figura 3.7.

No entanto, em alguns casos o tamanho do *data field* do APDU de resposta pode ser zero. Isso acontece quando o comando APDU for processado com erros ou enviar parâmetros incorretos. Neste caso essa situação é indicada pelos campos SW1 e SW2 do *trailler* (RANKL e EFFING, 2003, pág. 424). A figura 3.7 ilustra a estrutura básica de um APDU de resposta.

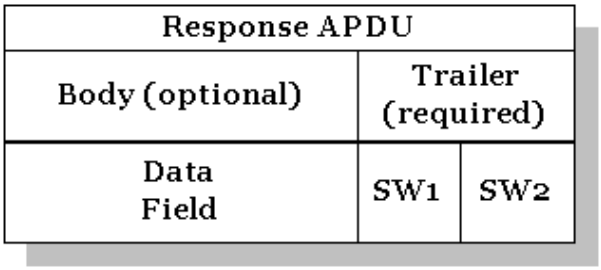

Figura 3.7 – Estrutura básica de um APDU de resposta.
Fonte: ORTIZ (2003a).

Dada a não obrigatoriedade do *data field* em um APDU de resposta, existem dois tipos de estruturas possíveis de respostas APDUs, ilustradas na figura 3.8.

Figura 3.8 – Tipos de estrutura de um APDU de resposta.
Fonte: RANKL e EFFING (2003, pág. 424).

O *smart card* sempre envia um *trailler* de resposta a um comando APDU. Os dois *bytes*, SW1 e SW2, que são chamados de códigos de retorno, codificam uma resposta ao comando. Por exemplo, o código de retorno ‘9000’ significa que o comando foi executado por

completo e com sucesso. Existem mais de 50 tipos de códigos de retorno diferentes. O esquema apresentado na figura 3.9 ilustra a classificação dos códigos de retorno segundo a ISO/IEC 7816-4 e GSM 11.11.

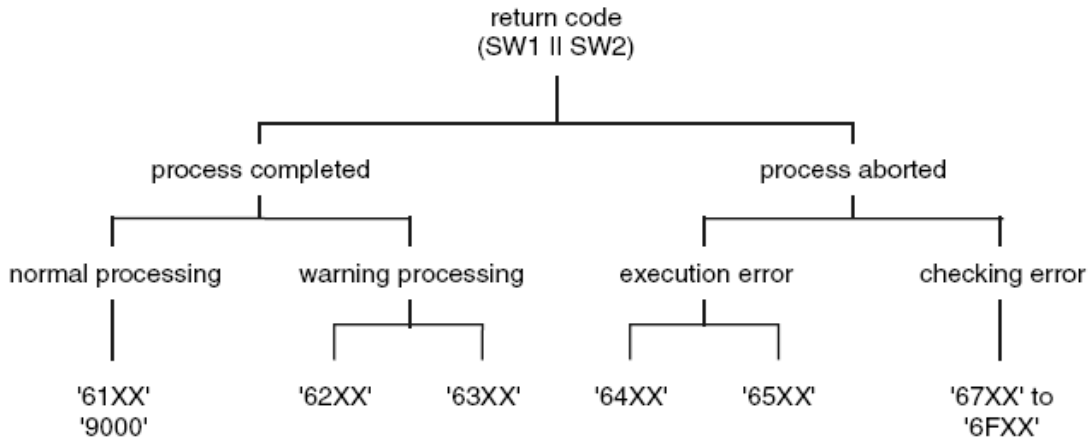

Figura 3.9 – Classificação dos códigos de retorno segundo a ISO/IEC 7816-4 e GSM 11.11.
Fonte: RANKL e EFFING (2003, pág. 425).

Se um APDU de resposta envia um código ‘63XX’ ou ‘65XX’ significa que a memória não-volátil do cartão, normalmente uma memória EEPROM, foi modifica. Já, se outro código começando com ‘6X’ é retornado, significa que a execução do comando foi encerrada prematuramente sem modificar o conteúdo da memória não-volátil (RANKL e EFFING, 2003, pág. 426).

3.11 *Java Card* e aplicações J2ME

A plataforma J2ME fornece um ambiente robusto e flexível para o desenvolvimento e execução de aplicações Java em dispositivos como telefones celulares, PDAs, TV *set-top boxes* e impressoras (SUN MICROSYSTEMS[17]).

A partir da versão 2.5.2 do Sun Wireless Toolkit for CLDC, a Sun Microsystems tornou possível a integração de aplicações J2ME com *smart cards* através das APIs *Security and Trust Service* (SATSA). O principal objetivo da especificação SATSA é fazer com que

[17] http://java.sun.com/javame/index.jsp. Acesso em 21 jun. de 2008.

aplicações J2ME possam ser utilizadas como aplicações *host* para *applets Java Card*. Assim, aplicativos J2ME executados em aparelhos celulares passaram a poder interagir com SIM *cards* adicionando valor e segurança para essas aplicações.

Da mesma forma como descrito anteriormente, a comunicação entre uma aplicação *host* J2ME e um *applet Java Card* utiliza um modelo síncrono de pedido/resposta (protocolo APDU), onde a aplicação J2ME é o cliente e o *applet Java Card* é o servidor, conforme ilustra a figura 3.10.

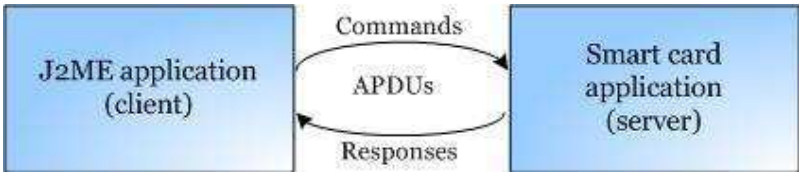

Figura 3.10 – Modelo básico de comunicação entre uma aplicação J2ME e um *smart card.*
Fonte: ORTIZ (2005).

3.12 JSR 177 - SATSA – Security and Trust Service APIs para J2ME

A *Security and Trust Service APIs* (SATSA) é uma especificação definida pela JSR 177 (*Java Specification Requests*). Ela define quatro pacotes que possibilitam que aplicações J2ME acessem e se comuniquem com dispositivos como *smart cards*. Ela também provê capacidades criptográficas a aplicações executadas nestes dispositivos. Segundo a Sun Microsystems (2004, pág. 7), as APIs SATSA podem permitir que aplicações MIDP[18] (*Mobile Information Device Profile*), executadas em telefones celulares GSM, por exemplo, acessem e interajam com aplicações executadas em SIM *cards*.

A especificação SATSA define pacotes opcionais que provêem APIs de segurança e confiança para aplicações J2ME, permitindo o acesso a serviços providos por um dispositivo seguro, como um *smart card*, como armazenamento seguro e leitura de informações sensíveis, bem como encriptações e serviços de autenticação (ORTIZ, 2003a).

[18] MIDP define uma arquitetura e APIs para o desenvolvimento e execução de aplicações executadas em dispositivos CLDC (Connect Limited Device Configuration), como telefones celulares e PDAs.

As quatro APIs definidas pela especificação SATSA são as seguintes:

a) **SATSA-APDU *Optional Package*** – permite que aplicações MIDP comuniquem-se com aplicações *smart cards* através de um protocolo de baixo nível denominado, simplesmente, APDU. Este protocolo é definido pela ISO-7816-4 e descrito na documentação do *Java Card* Development Kit;

b) **SATSA-JCRMI *Optional Package*** – provê um método alternativo para que aplicações comuniquem-se com aplicações *smart card* utilizando um protocolo de objeto distribuído remoto;

c) **SATSA-PKI *Optional Package*** – permite que aplicações utilizem *smart cards* para assinatura digital e gerenciamento de certificados digitais;

d) **SATSA-CRYPTO *Optional Package*** – provê suporte a funções de criptografia, como cifragens e assinaturas digitais.

Apesar de estarem definidas separadamente, funcionalmente não há diferenças entre as APIs SATSA-APDU e SATSA-JCRMI. Ambas provêem um meio de comunicação entre aplicações MIDP e o *smart cards* através de requisições da aplicação MIDP e recebimento de respostas do cartão. A diferença entre as duas APIs está na forma de implementação. A API SATSA-APDU utiliza um protocolo de baixo nível baseado na troca de *arrays* de *bytes* entre a aplicação MIDP e a aplicação *smart card*, enquanto a API SATSA-JCRMI apresenta um protocolo orientado a objeto, permitindo que a aplicação MIDP faça requisições ao *smart card* utilizando um objeto remoto (SUN MICROSYSTEMS, 2004, pág. 8).

4 BLUETOOTH

A seguir são apresentados os principais conceitos a cerca da especificação *Bluetooth* e que servem de embasamento teórico para o desenvolvimento do protótipo alvo deste trabalho. São abordados tópicos como o que é *Bluetooth*, pilha de protocolos *Bluetooth*, *Piconets* e *Scaternets*, segurança em *Bluetooth* e a API JSR 82.

4.1 A tecnologia sem fios *Bluetooth*

Para Thompson, Kline e Kumar (2008, p. 3), a tecnologia sem fios *Bluetooth* é uma especificação aberta de baixo custo, baixa potência, que opera através de ondas de rádio freqüência de baixo alcance permitindo conexões *ad hoc* sem fios entre dispositivos, seja para transmissão de voz e/ou dados. Por tratar-se de rádio freqüência de baixo alcance, as conexões atingem a distância de cerca de 10 metros, podendo chegar a 100 metros caso sejam utilizados transmissores de maior potência. Como normalmente opera com ondas de baixo alcance e potência, a especificação *Bluetooth* é adequada a dispositivos com fonte de energia baseada em bateria. Sua operação se dá na banda ISM (*Industrial, Scientific and Medical*), na faixa de freqüências entre 2,4 GHz e 2,4835 GHz. Esta faixa de freqüências está disponível em quase todo o mundo e não necessita de autorização para ser utilizada.

A especificação *Bluetooth* foi originalmente desenvolvida com o propósito de substituir os cabos até então utilizados nas conexões entre dispositivos, como telefones celulares, notebooks, PDAs, impressoras, mouses, teclados, fones de ouvido, entre outros. Entretanto, a conexão sem fio entre esses dispositivos permitiu a criação de outros cenários que não somente aquele da substituição dos cabos, como, por exemplo, a criação de redes pessoais. Desenhada como um sistema de redes sem fio de baixo custo para todas as classes de dispositivos móveis, a especificação *Bluetooth* tem a capacidade de facilmente criar redes

ad hocs entre esses dispositivos. A figura 4.1 ilustra diferentes cenários de aplicação da especificação *Bluetooth*, como a transmissão de voz e dados onde um dispositivo pode operar como ponto de acesso a um outro tipo de serviço ou à Internet, a substituição de cabos entre um *notebook* e seus periféricos e a criação de uma rede pessoal *ad hoc*.

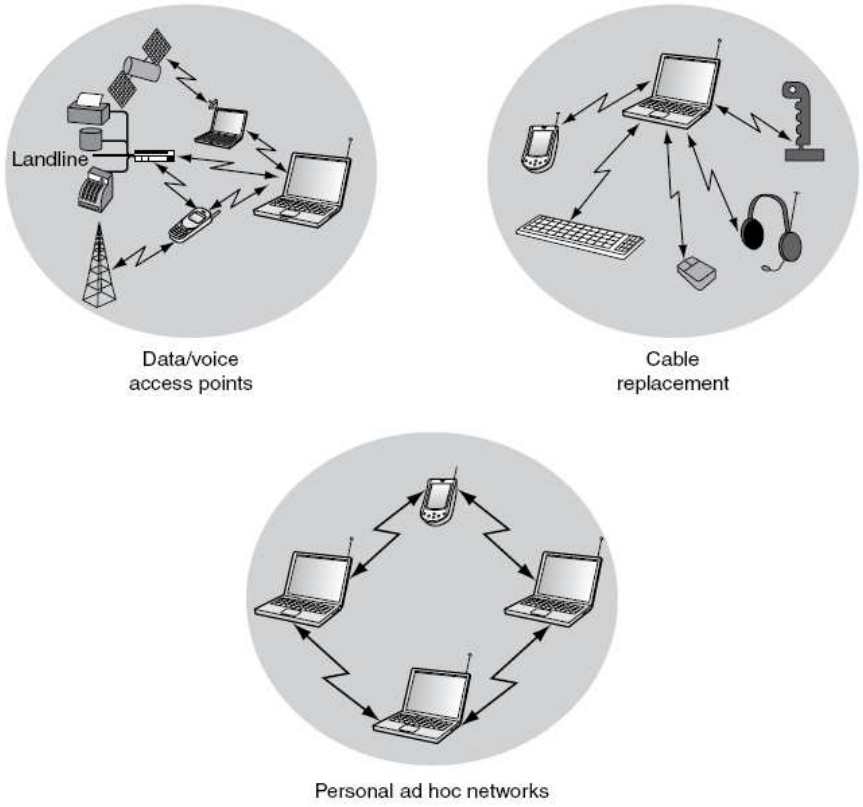

Figura 4.1 – Cenários de aplicações da especificação *Bluetooth*.
Fonte: Thompson, Kline e Kumar, 2008, p. 6.

4.2 A especificação *Bluetooth*

A especificação *Bluetooth* é o resultado da cooperação de diversas companhias líderes no segmento de tecnologia e comunicações, atuando através do consórcio *Bluetooth* SIG (*Special Interest Group*), cujos principais documentos estão publicados no *web site www.Bluetooth.com*. Ela define um comportamento para garantir a interoperabilidade entre diferentes fabricantes, além de um sistema completo que abrange desde as ondas de rádio até a camada de aplicação.

Analisando-se de um nível mais alto, a especificação *Bluetooth* é um conjunto de especificações assim composta: *Bluetooth Core Specification, Bluetooth Transport Specifications, Bluetooth Protocol Specifications* e *Bluetooth Profile Specifications*.

A *Bluetooth Core Specification* define a arquitetura global *Bluetooth*, termos, nomenclaturas e especificações para controladores *Bluetooth* e dispositivos *host*. A *Bluetooth Transport Specification* descreve como diferentes meios podem ser utilizados para comunicação entre os controladores e os dispositivos *host* (USB, PCMCIA, UART). A *Bluetooth Protocol Specification* descreve protocolos de alto nível que são executados sobre a tecnologia *Bluetooth* (RFCOM e OBEX). Por fim, a *Bluetooth Profile Specification* são especificações que descrevem perfis individuais *Bluetooth* (THOMPSON, KLINE E KUMAR, 2008, p. 7). Perfis *Bluetooth* descrevem um conjunto de capacidades das camadas de protocolos que representam um padrão de solução para um determinado modelo de aplicação.

4.3 A pilha de protocolos *Bluetooth*

Segundo Thompson, Kline e Kumar (2008, p. 8) a pilha de protocolos *Bluetooth* pode ser dividida em dois componentes: o *host Bluetooth* (*notebook*, telefone celular) e o controlador *Bluetooth* (ou módulo de rádio *Bluetooth*). Ainda, o *Host Controller Interface* (HCI) provê uma interface padrão entre esses dois componentes.

O *host Bluetooth* normalmente é implementado em software ou no sistema operacional do hardware *host*, que pode ser um *notebook* ou um telefone celular. Já o controlador *Bluetooth*, ou módulo de rádio, é um hardware implantado em PC *cards* ou dispositivos móveis como PDAs e *notebooks*. Esses dois componentes interagem através do HCI e normalmente utilizam interfaces de entrada e saída de dados como USB ou UART. Entretanto, em muitos dispositivos o *host* e o controlador *Bluetooth* estão integrados no mesmo hardware e não utilizam a interface HCI, como, por exemplo, os telefones celulares. A figura 4.2 ilustra a pilha de protocolos *Bluetooth*, com seus respectivos componentes a que normalmente são associados, além do HCI.

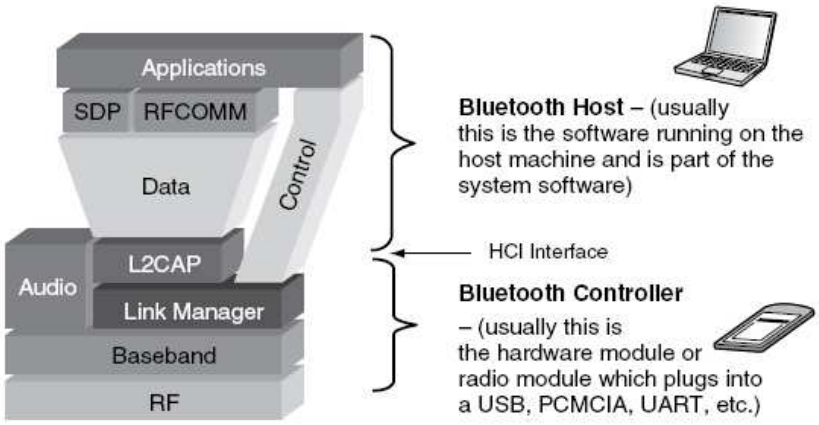

Figura 4.2 – Pilha de protocolos *Bluetooth* e seus respectivos componentes de hardware.
Fonte: Thompson, Kline e Kumar, 2008, p. 8.

A especificação *Bluetooth* define uma grande quantidade de protocolos, entretanto, a figura 4.3 ilustra aqueles considerados mais comuns, os quais são comentados a seguir. As caixas levemente sombreadas indicam protocolos utilizados pela *Java APIs for Bluetooth Wireless Technology* (JABWT).

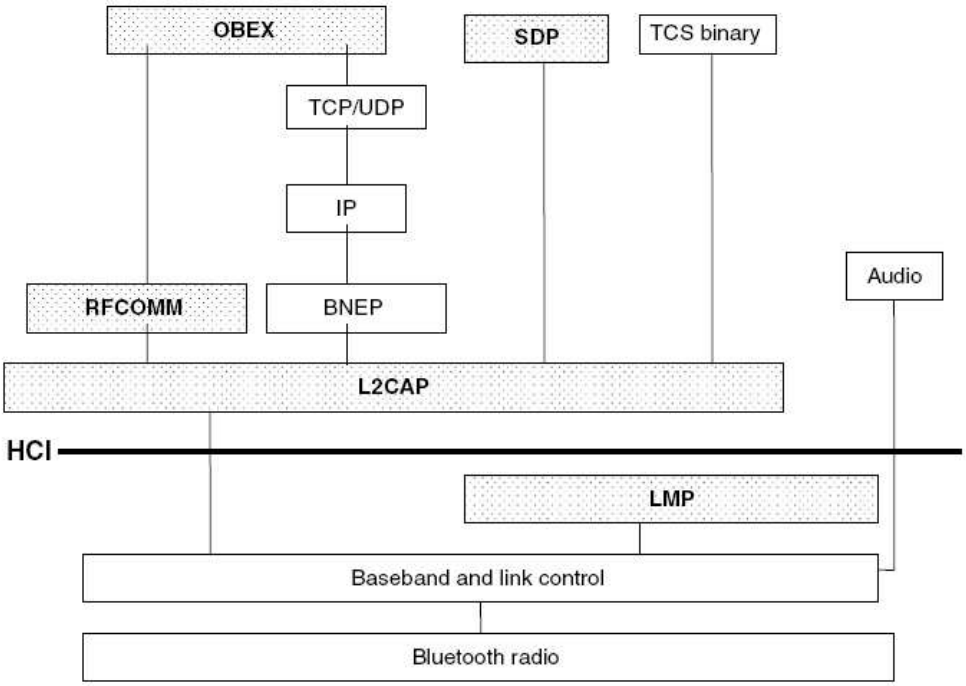

Figura 4.3 – Pilha de protocolos *Bluetooth*.
Fonte: Thompson, Kline e Kumar, 2008, p. 9.

Conforme ilustra a figura 4.3, os principais protocolos da pilha *Bluetooth* são os seguintes:

a) ***Bluetooth Radio***: a camada de rádio é a camada mais baixa definida pela especificação *Bluetooth*. Ela define os requisitos do dispositivo transceptor, dentre os quais a freqüência de operação de 2,4 GHz na banda ISM, o uso do *hopping* (salto de canais), esquema de modulação e o alcance da transmissão.

b) ***Baseband and Link Control***: esta camada provê o link físico de radio freqüência (RF) entre os dois dispositivos *Bluetooth* que estabelecem uma comunicação.

c) ***Audio***: esta não chega a ser uma camada da pilha de protocolos *Bluetooth*, mas é mostrada na figura, pois ele é tratado na comunicação *Bluetooth*. Dados de áudio são diretamente roteados da camada Baseband.

d) ***Link Manager Protocol* (LMP)**: esta camada é responsável pela configuração do link entre dois dispositivos *Bluetooth*, negociando e gerenciando o tamanho dos pacotes da camada *Baseband*. O LMP gerencia aspectos de segurança, tais como autenticação e encriptação, através da troca e verificação de chaves de criptografia.

e) ***Host Controller Interface* (HCI)**: provê uma interface de comando para o controlador da RF o gerenciador do link. É uma simples interface padrão para acesso a da camada *Baseband*, situação do hardware e controle de registros.

f) ***Logical Link and Adaptation Protocol* (L2CAP)**: esta camada oculta às camadas superiores os detalhes das camadas mais baixas. Organiza as diversas conexões lógicas feitas pelas camadas superiores.

g) ***Service Discovery Protocol* (SDP)**: é protocolo construído acima da camada L2CAP e fornece às aplicações um meio de buscar por serviços e características de conexões *Bluetooth*.

h) **RFCOMM**: este protocolo provê a emulação de portas serias sobre o protocolo L2CAP, proporcionando capacidades de transporte para serviços de níveis mais alto que utilizam uma interface serial como mecanismo de transporte. O RFCOMM ainda permite que dispositivos possuam múltiplas conexões simultâneas.

i) ***Bluetooth Network Encapsulation Protocol* (BNEP)**: este é um protocolo opcional que encapsula pacotes de diversas outros protocolos de rede. Esses pacotes encapsulados são transportados diretamente sobre o protocolo L2CAP.

j) ***Telephony Control Protocol Specification, Binary* (TCS *binary*)**: este protocolo é construído sobre o protocolo L2CAP e define um controle do sinal de chamadas para estabelecimento de chamadas de voz e dados entre dispositivos *Bluetooth*.

Protocolos adotados em outras redes como o OBEX e o IP são implementados sobre algum dos protocolos citados anteriormente, como, por exemplo, o OBEX que é construído sobre o protocolo RFCOMM e o IP implementado sobre o protocolo BNEP. A *Bluetooth* SIG está definindo novos protocolos que devem ser implementados sobre os protocolos citados anteriormente, principalmente sobre o protocolo L2CAP. Exemplos desses novos protocolos são o *Audio/Video Control Transport Protocol* e o *Audio/Video Distribution Transport Protocol*.

4.4 Perfis *Bluetooth*

Adicionalmente à pilha de protocolos, a *Bluetooth* SIG definiu padrões de uso para os seus protocolos e suas funcionalidades, chamados Perfis *Bluetooth*. Em outras palavras, eles definem como diferentes partes da especificação *Bluetooth* podem ser usadas para um fim específico. Um perfil pode ser descrito como uma fatia vertical através da pilha de protocolos. Ela define as opções do protocolo que são necessárias para determinado Perfil. As dependências das camadas de protocolos e suas características podem variar. Dois diferentes Perfis podem usar um conjunto de diferentes protocolos, ou um conjunto diferente de características dentro de uma mesma camada da pilha de protocolos (Thompson, Kline e Kumar, 2008, p. 11).

Um dispositivo *Bluetooth* pode suportar um ou mais Perfis. Os quatro Perfis básicos, segundo Thompson, Kline e Kumar (2008, p. 12), são os seguintes:

a) ***Generic Access Profile* (GAP)**: o GAP é a base de todos os outros perfis. Estritamente falando, todos os perfis são baseados no GAP, pois ele define os procedimentos genéricos relacionados ao estabelecimento de conexões entre dois dispositivos, incluindo a descoberta de dispositivos *Bluetooth*, o gerenciamento e configuração do link e os procedimentos relacionados com a utilização de diferentes de níveis de segurança;

b) ***Serial Port Profile* (SPP)**: A SPP define os requisitos necessários para que dispositivos *Bluetooth* criem emulações de conexões seriais através do protocolo RFCOMM, permitindo que aplicações legadas substituam cabos seriais por conexões *Bluetooth*;

c) ***Service Discovery Application Profile* (SDAP)**: o SDAP descreve os protocolos e procedimentos necessários para a descoberta de serviços em outros dispositivos *Bluetooth.*

d) ***Generic Object Exchange Profile* (GOEP)**: este é um Perfil abstrato em que podem ser implementados Perfis concretos. Estes Perfis utilizam OBEX e definem todos os elementos necessários para suportar os modelos de uso do OBEX, como a transferência de arquivos e sincronização.

4.5 *Piconets* e *Scatternets*

Uma das grandes vantagens da especificação *Bluetooth* é a possibilidade que a mesma dá a dispositivos para criarem PANs (*Personal Area Network*) *ad hoc*. Estas PANs são compostas de até oito dispositivos, sendo um mestre e até sete escravos. A esta estrutura de dispositivos mestre e escravos dá-se o nome de *Piconet.* As *piconets* podem conectar-se entre si através de seus dispositivos, formando, assim, uma estrutura denominada *Scatternet.*

Segundo Kobayashi (2004), uma limitação dessa estrutura é que um dispositivo pode ser mestre em uma única *piconet* em um determinado momento, podendo ter ao seu comando somente sete dispositivos. Esta limitação se dá pelo fato do cabeçalho do pacote de dados permitir endereçamento de até três *bits.* Em ligações ponto-a-ponto o mesmo canal é compartilhado entre os dois dispositivos, enquanto em ligações multi-ponto o mesmo canal é compartilhado por até oito dispositivos. Todos os membros de uma *piconet* atuam sincronizados conforme o *clock* e seqüência de saltos (*hopping*) do dispositivo mestre. Numa *piconet* toda comunicação ocorre entre um dispositivo escravo e o dispositivo mestre, de forma que toda informação enviada pela rede é passada pelo mestre e não existe comunicação direta entre nós escravos. A figura 4.4 ilustra uma *scatternet* formada por diversas *piconets.*

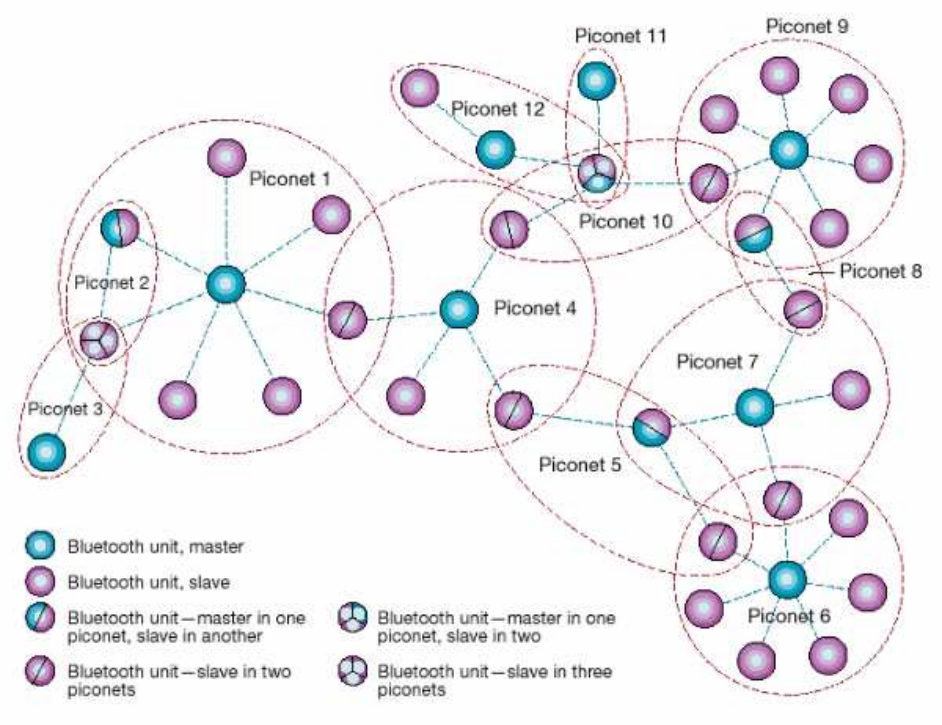

Figura 4.4 – *Piconets* formando uma *scatternet.*
Fonte: Frodigh, Johansson, Larsson, 2002.

4.6 Estrutura de rádio e pacotes *Bluetooth*

Conforme já citado, o sistema de rádio opera na freqüência de 2,4 GHz da banda ISM. A especificação *Bluetooth* define que um canal físico básico de *piconet* executa mudanças aleatórias nos 79 canais da freqüência, através de saltos chamados *hoppings*. São 1600 saltos por segundo, ou seja, uma troca de canal a cada 0,625 μseg, definidos pelo *clock* do dispositivo mestre. A cada salto de freqüência os transmissores e receptores são sintonizados ao mesmo tempo na nova freqüência.

Um pacote de dados *Bluetooth* é transmitido a cada 0,625 μseg, ou seja, no intervalo entre um salto de freqüência e outro. O pacote possui os campos Código de Acesso ao Canal, Cabeçalho do Pacote, Cabeçalho do *Payload*, *Payload* e CRC. Esse pacote é ilustrado na figura 4.5.

Código de Acesso ao Canal	Cabeçalho do Pacote	Cabeçalho do *Payload*	*Payload*	CRC

Figura 4.5 – Pacote de dados *Bluetooth*.
Fonte: Adaptado de *Bluetooth* SIG.

Ainda, o campo Cabeçalho do Pacote pode ser dividido conforme ilustrado na figura 4.6. É possível observar que o campo Endereço possui somente três bits, o que faz com que uma *piconet* possa possuir no máximo oito dispositivos *Bluetooth*.

Endereço	Tipo de Pacote	Controle de Fluxo	Confirmação de Bit	Controle de Seqüência	Checagem de erro
3 bits	4 bits	1 bit	1bit	1bit	8 bits

Figura 4.6 – Detalhamento do campo Cabeçalho do Pacote do pacote de dados *Bluetooth*.
Fonte: Adaptado de *Bluetooth* SIG.

4.7 Segurança

Conforme Mahmound (2003a), a segurança na especificação *Bluetooth* é fornecida de três maneiras: salto de freqüências (*hopping*), autenticação e encriptação. Os saltos de freqüência dificultam que outro dispositivo possa interceptar uma comunicação sem que esteja sincronizado com o Servidor, a autenticação permite a um usuário limitar a conectividade para dispositivos especificados, enquanto a encriptação utiliza chaves criptográficas para tornar os dados legíveis somente às partes autorizadas.

Todos os dispositivos *Bluetooth* qualificados devem implementar o perfil GAP, responsável por implementar os modos de segurança. A especificação *Bluetooth* disponibiliza três modos de segurança para acesso *Bluetooth* entre dois dispositivos e que cabem aos fabricantes determinar a sua implementação. São os seguintes modos:

a) **Sem segurança**: neste modo os dispositivos nunca irão iniciar um procedimento de segurança e a autenticação é opcional;

b) **Segurança em níveis de serviço**: neste modo a segurança é garantida em nível de serviço, ou seja, o serviço é responsável por optar pela utilização da segurança. Conforme Guisi (2007, p. 42), “Os procedimentos do modo de segurança 2 são iniciados pelas camadas superiores da pilha de protocolo *Bluetooth*, o que permite que desenvolvedores decidam para qual serviço será necessário suporte a segurança”. Adicionalmente, para dispositivos podem existir dois níveis: dispositivo confiável e dispositivo não confiável. Um dispositivo confiável é aquele que já foi emparelhado com algum outro dispositivo, e tem livre acesso a todos os serviços. Ainda, para os serviços, existem três diferentes níveis: serviços que requerem autenticação e autorização, serviços que requerem apenas autenticação e serviços abertos a todos os dispositivos (BLUETOOTH SIG).

c) **Segurança em nível de conexão**: neste modo os procedimentos de segurança são iniciados ao iniciar-se a configuração da conexão *Bluetooth*. Caso os procedimentos de segurança não tenham sucesso, a conexão não se efetivará. É um modo iniciado pelas camadas inferiores da pilha de protocolos *Bluetooth* e os desenvolvedores não podem configurá-la (GUISI, 2007, p. 42).

Devido ao escopo do presente trabalho, o detalhamento das questões de segurança relativas à especificação *Bluetooth* não serão aqui abordadas. Para aprofundamentos nessas questões recomenda-se a leitura da obra *Bluetooth Security*, dos autores Gehrmann, Persson e Smeets, editora *Artech House*, 2004 e SUN MICROSYSTEMS, 2008.

4.8 API JSR 82 – *Java APIs for Bluetooth*

A *Java APIs for Bluetooth* – JSR 82 é uma especificação definida pela *Community Development of Java Technology Specification* cujo foco principal é prover conexões *Bluetooth* entre pequenos dispositivos a partir de aplicações J2ME. Sua implementação é baseada no *Generic Connection Framework* (GCF) definido no J2ME *Connected Limited Device Configuration* CLDC. Entretanto, ela é especificada de forma a permitir que sejam adicionadas camadas que possibilitem o seu uso em outras plataformas Java como o CDC, J2SE e J2EE. A sua concepção consiste na implementação de pacotes: o *javax.bluetooth* e o

javax.obex, que provêem, respectivamente, o núcleo da API *Bluetooth* e a *Object Exchange* (OBEX) API.

Atualmente, conforme a SUN MICROSYSTEMS (2008), a API JSR 82 não contempla todas as áreas definidas pela especificação *Bluetooth*, de forma que ação abordadas as seguintes áreas: transmissões de dados apenas (áudio não é suportado), os protocolos L2CAP (somente orientado a conexões às camadas superiores), RFCOMM, SDP (*Service Discory Protocol*), OBEX (*Object Protocol Exchange*), e os perfis GAP (*Generic Access Profile*), (SDAP) *Service Discovery Application Profile*, SPP (*Serial Port Profile*), GOEP (*Generic Object Exchange Profile*). Quanto às áreas não abordadas, estão transmissão de audio e o protocolo *Telephony Control Protocol – Binary* (TCS *Binary*)

5 MODELO PROPOSTO E PROTOTIPAÇÃO

Neste capítulo é apresentado e contextualizado o modelo e protótipo colocado como objetivo geral deste trabalho, bem como a metodologia de pesquisa, ferramentas e ambientes de desenvolvimento utilizados no projeto. Este modelo é uma prova de conceito, onde o foco se manteve centrado na operacionalização de transferências eletrônicas através de uma comunicação *Bluetooth*, ponto a ponto, entre dois dispositivos móveis.

5.1 O Modelo Proposto

Imagine-se a seguinte situação: um usuário do sistema carteira eletrônica dirige-se a uma tabacaria. Tão logo entra na loja, ele visualiza os charutos cubanos que mais agradam o seu paladar e que a muito tempo procurava. Porém, o mesmo está sem sua carteira física e, conseqüentemente, sem cartões de crédito, cartões de débito ou dinheiro em espécie. Para sua sorte, em seu bolso encontra-se o seu aparelho celular GSM, cujo SIM *card* possui uma carteira eletrônica com saldo. Por tratar-se de sistema altamente difundido e aceito, o comerciante também possui este sistema para pagamento por suas mercadorias.

Assim, após ambos acordarem a forma de pagamento, empunham seus aparelhos celulares e inicializam a aplicação Carteira Eletrônica, obviamente já instalada em seus dispositivos. Através da aplicação que faz uso da tecnologia *Bluetooth*, é estabelecida uma conexão ponto a ponto entre os dois aparelhos celulares. O comprador, visualizando a aplicação na tela do seu aparelho celular, informa o valor do pagamento e autoriza a transferência de valores da sua carteira eletrônica para a do vendedor através da digitação da sua senha. Esta operação gera comandos APDUs da aplicação *host* para o *applet Java Card* residente no SIM *card* do aparelho celular do comprador. O *applet* realiza suas validações, efetua o débito no saldo do comprador e envia uma resposta APDU ao *host* autorizando o

crédito daquele valor ao vendedor. Esta informação é então encapsulada e transmitida à aplicação *host* do vendedor através da conexão *Bluetooth* anteriormente estabelecida. Ao receber a informação a aplicação *host* do vendedor a transfere ao seu SIM *card* que realiza as suas validações e credita o saldo à carteira eletrônica, retornando uma resposta de confirmação ao vendedor e, através da aplicação *host* e da conexão *Bluetooth*, também ao comprador. Uma vez confirmando o pagamento, comprador e vendedor finalizam as aplicações J2ME em seus aparelhos celulares e o comprador deixa a loja degustando um de seus charutos cubanos.

Posteriormente o vendedor solicita o reembolso dos valores recebidos eletronicamente para dinheiro em espécie junto à entidade operadora do sistema de carteira eletrônica. Analogicamente, um comprador supre sua carteira eletrônica através da troca de dinheiro em espécie por dinheiro eletrônico, também junto à entidade operadora do sistema.

A situação descrita anteriormente pode ser reproduzida em inúmeras outras transações comerciais, como o pagamento de corridas de táxi, tele-pizzas, restaurantes, prestadores de serviço, entre muitas outros. A arquitetura geral do modelo é apresentada na figura 5.1.

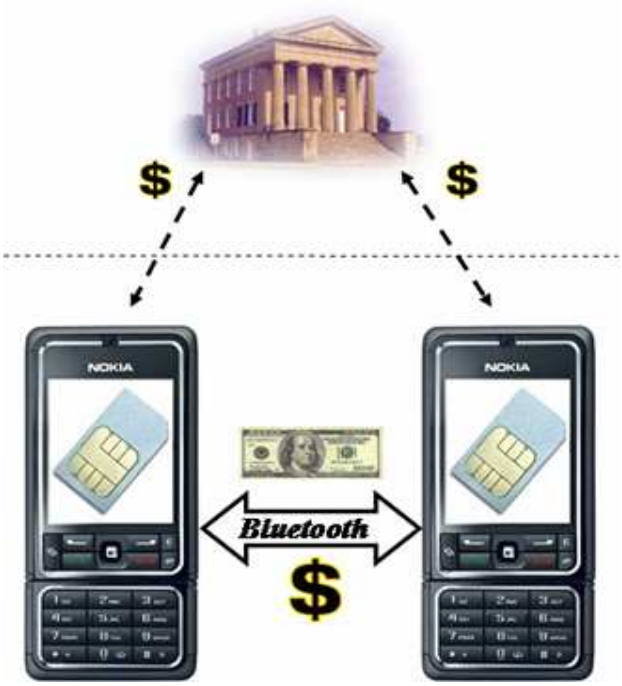

Figura 5.1 – Estrutura geral do modelo proposto.
Fonte: Elaborada pelo autor.

É importante salientar que estas transações ocorrem de forma *off-line*, ou seja, sem a participação da entidade operadora do sistema de carteira eletrônica para validá-las.

5.2 Tecnologias Envolvidas

O modelo acima apresentado tem sua estrutura baseada e sustentada pelas seguintes tecnologias:

a) ***Smart card*** – a tecnologia *smart card* será utilizada para armazenar o *applet Java Card* responsável por realizar as operações de débito e crédito da carteira eletrônica e guarda o saldo daquele usuário. As características nativas de segurança dos *smart cards*, proporcionadas pelos JCRE e JCVM e aplicações *card manager* e WIM foram determinantes na opção por estes dispositivos;

b) ***Java Card*** – a plataforma *Java Card* será utilizada no desenvolvimento do *applet* que será carregado nos *smart cards*. A disponibilidade de acesso à plataforma, o conhecimento prévio da linguagem Java, bem como todas as demais características já citadas neste trabalho foram decisivas na opção por esta plataforma. Para o desenvolvimento do *applet Java Card* será utilizado a IDE de desenvolvimento Eclipse, juntamente com o plug-in *JCOP Tools* e as APIs *Java Card*. Para a realização de testes também pretende-se utilizar o aplicativo *Smart card Toolset PRO v3.3.5* da empresa *SCard Soft*;

c) **J2ME** – a plataforma J2ME será utilizada no desenvolvimento da aplicação *host* que deverá ser instalada nos dispositivos móveis, especificamente em aparelhos celulares. Esta aplicação será responsável por estabelecer o canal de comunicação com o *applet Java Card* residente no *smart card* – SIM *card* – através do envio e recebimento de comandos e respostas APDUs. A aplicação *host* também é responsável por fornecer uma interface amigável aos usuários, permitindo o fácil acesso às funções de transferência de valores, consultas de saldo e carga e descarga de dinheiro eletrônico. Para o desenvolvimento, testes e simulação serão utilizadas as ferramentas IDE *NetBeans 6.0.1* e *Wireless Toolkit 2.5.2 for CLDC*, ambas da Sun Microsystems, juntamente com as APIs JABWT;

d) ***Bluetooth*** – a tecnologia *Bluetooth* será responsável por permitir a conexão *wireless* entre os dois aparelhos celulares. Características como custo zero de comunicação, baixo consumo de energia, grande disseminação entre os dispositivos móveis atuais, uma vez que grande parte destes dispositivos possuem uma interface *Bluetooth*, aliados a facilidade de utilização e a possibilidade de integração com aplicações J2ME através das API JABWT, foram determinantes na sua escolha.

5.3 Considerações sobre o Protótipo

Inicialmente pretendia-se que o protótipo desenvolvido fosse fisicamente implementado e testado utilizando SIM *cards* ou *smart cards* e aparelhos celulares. Porém, tendo em vista as dificuldades e limitações encontradas no decorrer do desenvolvimento do projeto, a implementação e validação do protótipo foram direcionadas para um ambiente de emulação, utilizando ferramentas que serão descritas adiante. Tais dificuldades e limitações serão abordadas no capítulo final deste trabalho.

Dada a complexidade natural de um sistema de pagamento eletrônico seguro e a limitação de tempo ao qual este trabalho está sujeito, o escopo do projeto ficou restrito a prova de conceito envolvendo a transferência de valores monetários armazenados em *smart cards* através de uma comunicação ponto a ponto entre dois aparelhos celulares, via *Bluetooth*.

Desta forma o modelo de carteira eletrônica proposto não contemplou funções de segurança como o uso de certificados digitais e assinatura dos pacotes ou geração de chaves criptográficas randômicas para cada transação. Todas essas funções de segurança que podem tornar o modelo viável deverão ser implementadas em trabalhos futuros.

5.4 Ambiente de Desenvolvimento

O desenvolvimento prático do protótipo proposto compreendeu a codificação, teste, execução e simulação do *applet Java Card* e do MIDlet J2ME. Para tanto foram utilizadas ferramentas e ambientes de desenvolvimento, teste e simulação para a linguagem Java e as APIs J2ME e *Java Card*, tais como o IDE NetBeans 6.0.1, IDE Eclipse 3.4 e IBM JCOP 3.1.2, *Sun Wireless Toolkit 2.5.2* e *C-Language Java Card RE - cref*. A seguir os ambientes utilizados são brevemente descritos.

5.4.1 NetBeans 6.0.1

O desenvolvimento do MIDlet *CarteiraBluetoothMIDlet* foi realizado através da IDE (*Integrated Development Environment*) NetBeans 6.0.1. Esta ferramenta proporciona um ambiente completo para criar, testar, depurar e implementar aplicações para dispositivos móveis como telefones celulares, PDAs, *set up boxes*, sistemas embarcados (*embedded systems*) que possuem uma JVM instalada. Possui uma grande gama de bibliotecas e APIs que permitem a criação de aplicações para *Mobile Information Device Profile* (MIDP) 1.0, 2.0, 2.1 (MSA), *Connected Limited Device Configuration* (CLDC) 1.0 e 1.1, e *Connected Device Configuration* (CDC). O ambiente de teste compreende simulação serviços de conexão web, interfaces *wireless* (API JSR 82 - *Bluetooth*), interface com dispositivos seguros (simulação de SIM *cards* – API JSR 177), envio de mensagens SMS, emulação de *Over-the-Air* (OTA), assinatura de MIDlets, gerenciamento de certificados, entre outros. A figura 5.2 ilustra o ambiente de desenvolvimento NetBeans 6.0.1.

Devido ao foco deste trabalho, a IDE NetBeans não será explorada aqui. Para detalhes, recomenda-se a leitura da documentação disponível em *http://www.netbeans.org*.

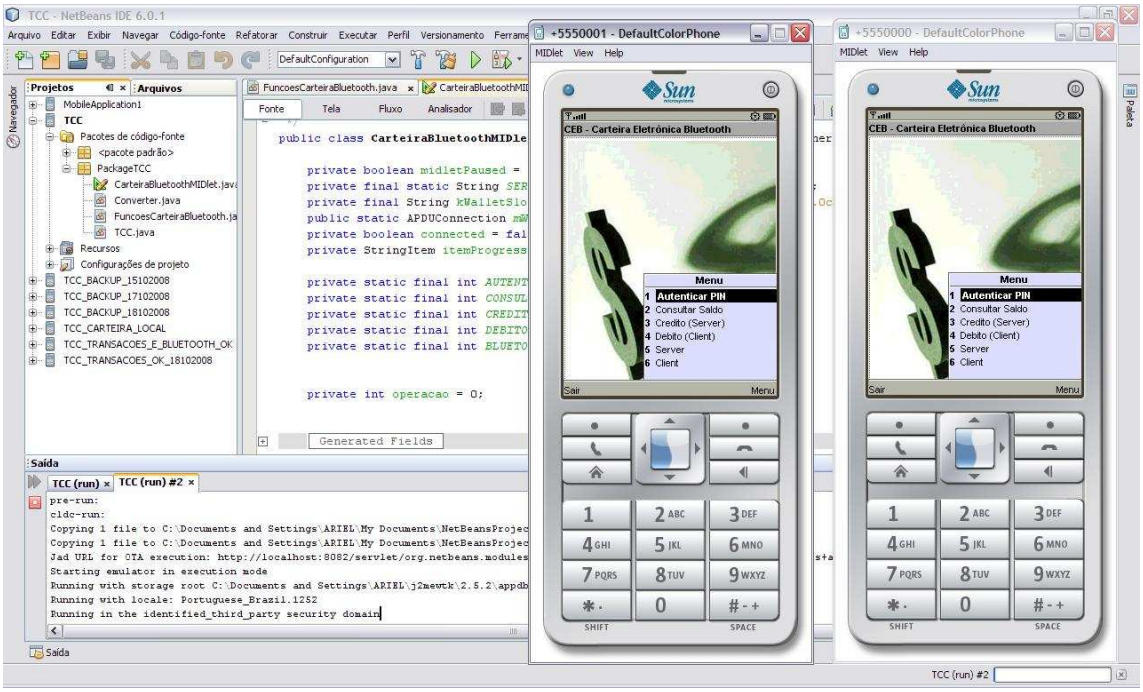

Figura 5.2 – Ambiente de desenvolvimento NetBeans 6.0.1.
Fonte: Elaborada pelo autor.

5.4.2 IBM JCOP 3.1.2

O IBM JCOP Tools 3.1.2 é um *plugin* que oferece um conjunto de ferramentas para o desenvolvimento, teste e implantação de aplicações para dispositivos *Java Card OpenPlatform*. O JCOP é uma implementação das principais especificações *Java Card 2.2.1*[19] e *Global Platform Consortium: OpenPlatform 2.1.1*[20], incluindo alguns refinamentos da *Visa OpenPlatform Card Implementation Requirements*[21]. Ele oferece um conjunto de linhas de comandos (console) associados com um ambiente gráfico de desenvolvimento que permite a criação, modificação e gerenciamento dos projetos e seus códigos fontes, simplificando a detecção de erros e resolução de problemas durante a compilação e processo de conversão dos *applets*, além de fornecer um ambiente de simulação, teste e depuração em nível de código para as aplicações *Java Card* desenvolvidas.

Ainda, o JCOP fornece todas as ferramentas e scripts para simplificar e facilitar a carga, instalação e teste de *applets Java Card* em *smart cards* reais *compliance* com as especificações *Java Card 2.2.1* e *OpenPlatform 2.1.1*. Possui suporte para leitoras PC/SC o que lhe garante compatibilidade com uma grande gama de leitoras e terminais com e sem contato (*JCOP Tools 3.0 (Eclipse Plugin) – Technical Brief*). O *plugin* IBM JCOP ainda possui portabilidade, suportando sistemas operacionais Windows, Linux e MAC OS.

5.4.2.1 Ambiente de simulação JCOP Tools

O IBM JCOP Tools é distribuído com simulações dos mais importantes *smart cards* membros da família JCOP, tais como JCOP 10, JCOP 11, JCOP 20, JCOP 21, JCOP 30, JCOP 31 e JCOP 41. Estas simulações são executadas na estação de desenvolvimento (*host*), apresentando comportamento muito semelhante aos cartões JCOP reais. As simulações fornecem as mesmas limitações de memórias RAM, EEPROM e ROM dos cartões físicos, além das velocidades limitadas de execução dos chips reais.

Assim, *applets* podem ser carregados, instalados e executados em um ambiente de simulação que permite o acompanhamento do progresso das execuções através das ferramentas de depuração.

---

[19] <http://java.sun.com/javacard/>
[20] <http://www.globalplatform.org/>
[21] <http://www.visa.com/nt/suppliers/vendor/>

5.4.2.2 Console (JCOP Shell)

O JCOP *Tools* também oferece uma console que pode ser utilizada tanto de forma *on-line* como em modo *batch*. Esta última opção permite executar *scripts* de testes complexos. Ainda, o console não só permite o envio e recebimento de comandos APDUs, mas também oferece um grande conjunto de comandos para manipulação dos dados do *smart card*, tais como a autenticação, carga e exclusão de pacotes, instalação, exclusão e seleção de *applets*, entre outros.

As figuras 5.3 e 5.4 apresentam, respectivamente, as APIs que compõe o IBM JCOP e que devem ser adicionadas a um projeto e o ambiente de desenvolvimento e simulação do plugin IBM JCOP.

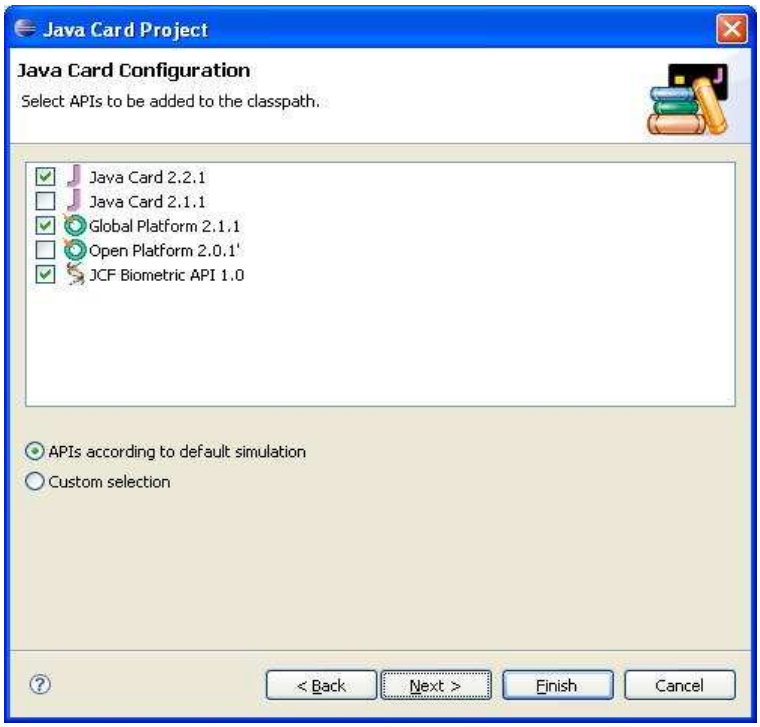

Figura 5.3 – Criação de um projeto *Java Card*. Seleção de APIs no *plugin* JCOP.
Fonte: Elaborada pelo autor.

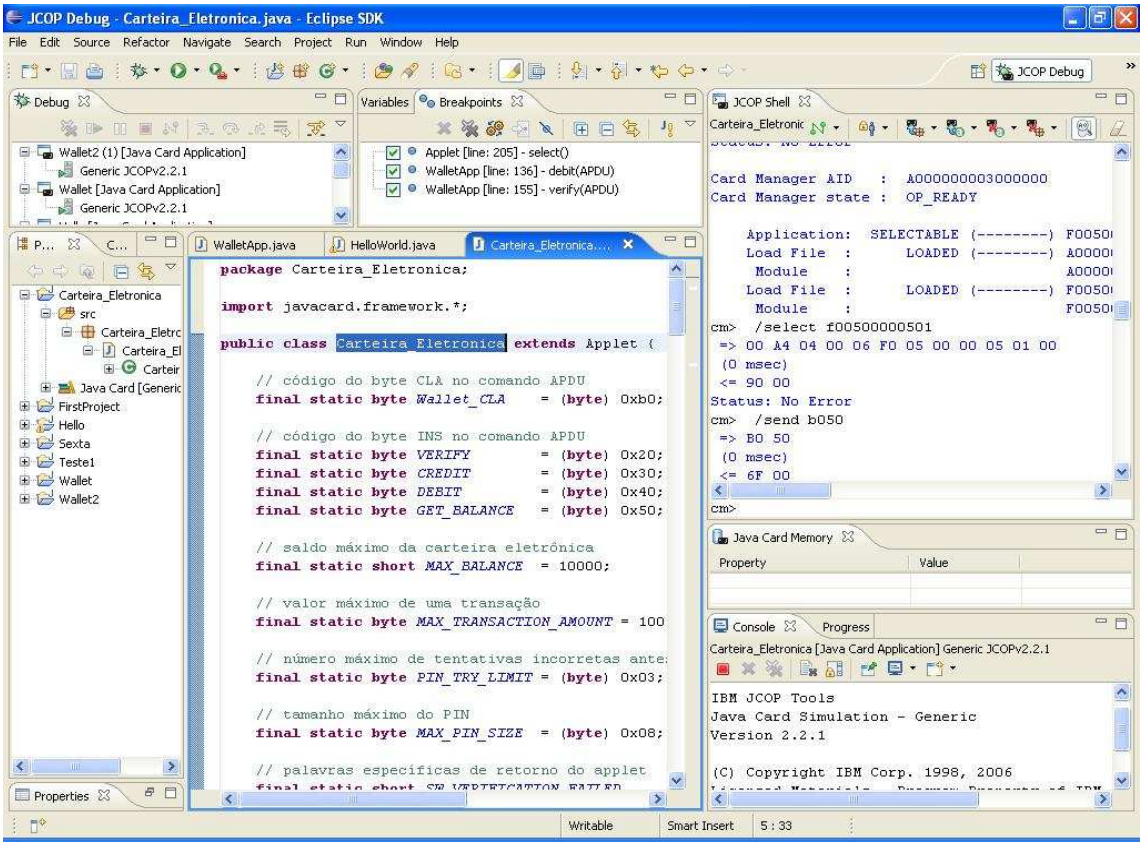

Figura 5.4 – Ambiente do *plugin* IBM JCOP na IDE Eclipse.
Fonte: Elaborada pelo autor.

5.4.3 Hardware de teste – Cartões JCOP e leitora CAD

Para a validação das funcionalidades do *plugin* IBM JCOP foram adquiridos alguns *smart cards* NXP JCOP 21 V2.2 72KB junto a empresa *UsaSmartCard*, localizada na cidade de Nova York, EUA, acessível através do *web site http://www.usasmartcard.com*. Os cartões adquiridos implementam uma JCVM versão 2.2 e possuem um chip EEPROM com capacidade de armazenamento de 72 KB.

Para a leitura e gravação dos *smart cards* foi adquirida uma leitora PC/SC modelo GemPCTwin, fabricado pela empresa Gemplus.

As figuras 5.5 e 5.6 ilustram, respectivamente, os cartões JCOP e a leitora PC/SC adquiridos.

Figura 5.5 – Cartão JCOP 21 V2.2 72 KB.
Fonte: Elaborada pelo autor.

Figura 5.6 – Leitora PC/SC GemPCTwin.
Fonte: Elaborada pelo autor.

5.4.4 Dificuldades encontradas

Inicialmente, mesmo que não explícito no escopo inicial do projeto, pretendia-se que a validação do protótipo desenvolvido fosse integralmente realizada em dispositivos móveis reais, especificamente em dois telefones celulares que atendessem aos requisitos mínimos de conectividade *Bluetooth* e implementação de APIs por sua JVM.

Para tanto, foram adquiridos *smart cards* JCOP, apresentados na seção anterior, para a instalação dos *applets* e uso nos telefones celulares. Conforme já relatado, tendo em vista que o acesso a essa tecnologia ainda é muito restrito a fabricantes de soluções e empresas de telefonia, a aquisição desses *smart cards Java Card* teve de ser feita através da importação dos mesmos a partir de uma empresa fornecedora localizada na cidade de *New York*, EUA. Cabe citar que por diversas vezes tentou-se a aquisição de SIM *cards* junto a grandes fabricantes localizados no estado de São Paulo, porém não foi obtido êxito em qualquer das tentativas realizadas.

Assim, após carga das aplicações nos *smart cards* JCOP, os mesmos foram recortados conforme as dimensões ID-000 para inserção nos telefones celulares. Entretanto, por motivo desconhecido, percebeu-se que todos os modelos que atendiam os requisitos de conectividade e implementação de APIs rejeitavam os *chips* adquiridos e não inicializavam suas funções. Até então, se pensava que todos os celulares que podem ser iniciados e ter suas funções de conectividade *Bluetooth* utilizadas sem a necessidade de existir um SIM *card* inserido, também aceitariam os *chips* JCOP.

Durante o desenvolvimento da pesquisa, alguns aparelhos foram testados, buscando-se identificar modelos que pudessem ser utilizados. Dentre os aparelhos celulares testados estão os modelos Nokia 5200, Nokia 5310, Nokia N73, Nokia E50, que são aparelhos cujas JVMs implementam as APIs necessárias ao protótipo. Entretanto, todos rejeitaram os *smart cards* JCOP. Buscando elucidar o problema, testou-se, também, um aparelho HTC que executa o sistema operacional Windows Mobile 6.0. Esse telefone aceitou o chip JCOP, porém, não atende aos requisitos de implementação das APIs JSR 177 e JSR 82. Outro aparelho testado foi o Motorola A1200e que executa um sistema operacional Linux. Esse aparelho também aceitou o *smart card* JCOP e executou com sucesso a aplicação de conexão *Bluetooth*. Porém, não atende o requisito de implementação da API JSR 177 para conexão com o *smart card*.

Assim, baseado no fato que os únicos telefones que aceitaram o *chip* JCOP são aqueles que executam sistemas operacionais Windows Mobile e Linux, enquanto aqueles que executam Symbian OS e Nokia OS não os aceitaram, concluiu-se que os sistemas operacionais executados pelos aparelhos celulares podem estar relacionados ao fato do dispositivo rejeitar ou não um *smart card* JCOP que não possui uma aplicação GSM (SIM *card*).

Cabe citar que a pesquisa por telefones que aceitassem o *smart card* JCOP encontrou limitações no que diz respeito ao acesso a estes aparelhos, principalmente a modelos recentes, considerados de última geração. Modelos que utilizam sistema operacional Linux ou Windows Mobile e que implementam as APIs necessárias poderiam apresentar resultados satisfatórios quanto a aceitação e operacionalização do *smart card* JCOP.

Portanto, dadas as dificuldades e restrições técnicas encontradas, optou-se por validar o protótipo em um ambiente totalmente simulado, por meio das ferramentas descritas neste capítulo.

5.5 Ambiente de Simulação

A validação do protótipo desenvolvido se deu através de simulação de um ambiente composto por dois dispositivos móveis (telefones celulares), dois *smart cards* e uma conexão *wireless Bluetooth* entre os dois telefones celulares. A seguir, as ferramentas utilizadas para execução da simulação são brevemente apresentadas.

5.5.1 Sun Java Wireless Toolkit 2.5.2 for CLDC

O *Sun Java Wireless Toolkit for CLDC* consiste em um conjunto de ferramentas e utilitários para construção de aplicações compatíveis com as especificações *Java Technology for the Wireless Industry* (JTWI, JSR 185) e *Mobile Service Architecture* (MAS, JSR 248), além de um emulador de telefones celulares utilizado para testar aplicações MIDP.

O *Java Wireless Toolkit for CLDC* suporta diversas APIs definidas pela *Java Community Process* (JCP), possibilitando a emulação de aplicações que utilizam tais APIs, dentre as quais a JSR 177 – SATSA, JSR 82 – *Bluetooth*, JSR 75 – PIM *and File*, JSR 185 – JTWI, JSR 172 – *Web Services*, JSR 248 – MAS, JSR 184 – 3D *Graphics*, JSR 229 – *Payment* API. A relação completa pode ser consultada em SUN MICROSYSTEMS (2007).

5.5.1.1 Utilizando SATSA no WTK 2.5.2

Dispositivos móveis que executam aplicações SATSA possuem um ou mais slots para *smart cards* (SIM *cards*). Para comunicar-se com *smart cards*, essas aplicações precisam especificar o slot do cartão e o *Application Identifier* (AID) da aplicação. Para a completa simulação de um telefone celular o WTK 2.5.2 possui um emulador que se comunica com aplicações *smart card* (*Java Card*) através de *sockets* TCP. A comunicação via *socket* pode ser estabelecida com um simulador de *smart cards* (*cref*) ou através de um *proxy* capaz de comunicar-se com cartões reais.

O emulador do *Sun Java Wireless Toolkit* 2.5.2 possui suporte completo para aplicações que utilizam a API JSR 177 – SATSA, através da emulação de dois slots para simuladores de *smart cards*. Cada slot está associado a um *socket* que representa uma comunicação com uma aplicação *smart card*. Através do menu preferências da ferramenta é possível configurar as portas TCP utilizadas pelos *sockets*. Por padrão, estão definidas as portas 9025 e 9026 para os slots 0 e 1, respectivamente. A figura 5.7 ilustra as configurações do simulador de slots de *smart cards* (SIM *cards*).

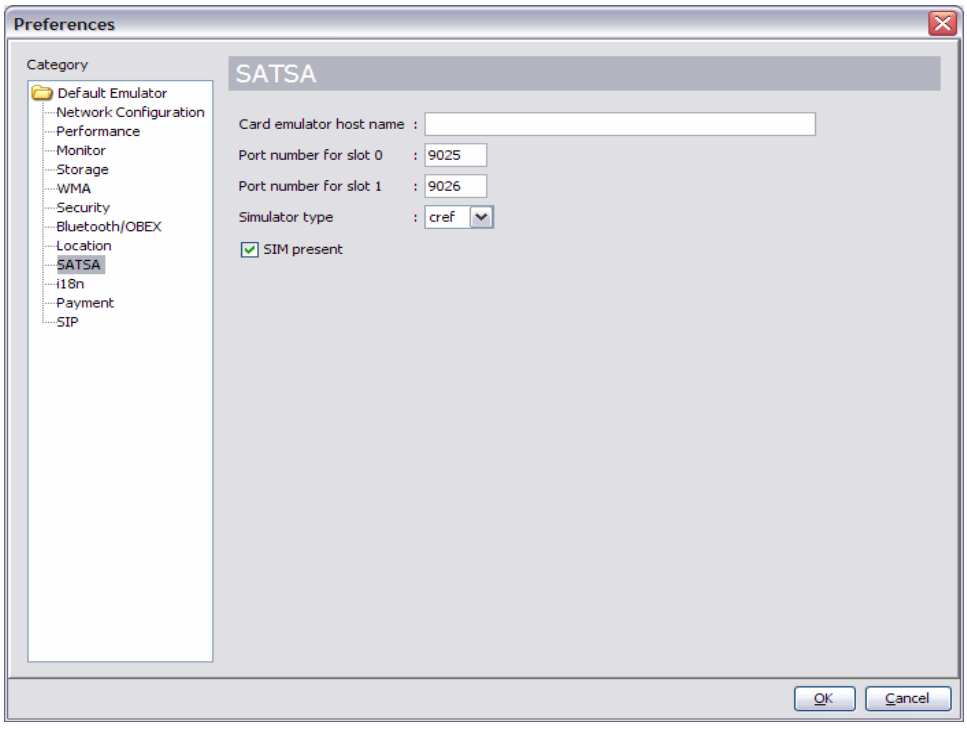

Figura 5.7 – Configurações do simulador SATSA do *Sun Java Wireless Toolkit 2.5.2*.
Fonte: Elaborada pelo autor.

Para o correto funcionamento é necessário que uma ou mais instâncias do *C-Language Java Card RE – cref* seja executada em paralelo ao WKT 2.5.2 para que seja estabelecido o *socket* de comunicação com o *applet Java Card* que é executado no simulador.

5.5.1.2 Ambiente de simulação *Bluetooth*

O *Sun Java Wireless Toolkit for CLDC* disponibiliza um ambiente para desenvolvimento e teste de aplicações MIDP que fazem uso da API JSR 82 – *Bluetooth*. Ele é capaz de simular um ambiente *Bluetooth* onde múltiplas instâncias do emulador são capazes de se comunicar umas com as outras.

Através do menu preferências do WTK é possível configurar determinadas propriedades dos dispositivos *Bluetooth* que podem ser acessadas em uma aplicação através do método *getProperty()* disponível na biblioteca *javax.Bluetooth.LocalDevice*. A figura 5.8 ilustra algumas propriedades configuráveis do WTK 2.5.2.

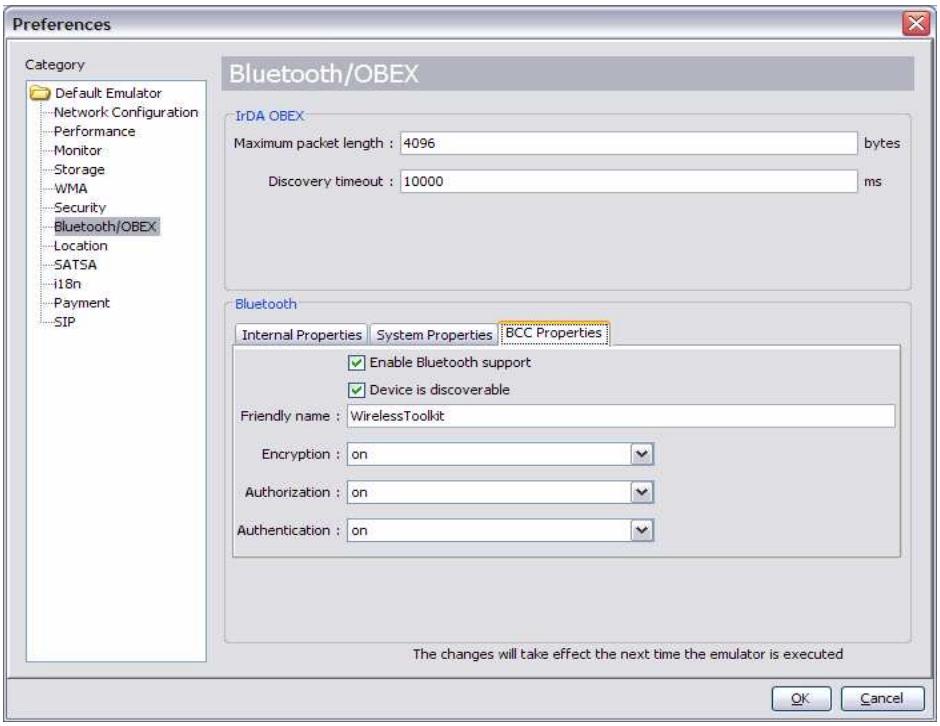

Figura 5.8 – Configurações do simulador *Bluetooth* do Sun Java Wireless Toolkit 2.5.2.
Fonte: Elaborada pelo autor.

Tendo em vista o foco do presente trabalho, não são apresentados aqui detalhes da ferramenta WTK 2.5.2. Para maiores informações recomenda-se a leitura de SUN MICROSYSTEMS, 2007.

5.5.2 *C-Language Java Card RE – cref*

O *Java Card Toolkit 2.2.2* traz consigo um simulador chamado *C-Language Java Card Runtime Environment* (*C-language Java Card RE*), ou *cref*, escrito em linguagem C, capaz de implementar uma plataforma *Java Card*. Este simulador é capaz de implementar uma plataforma *Java Card* e construir a imagem de uma memória ROM muito próxima de uma implementação real da tecnologia *Java Card*. Ele também é capaz de simular uma memória persistente (EEPROM), salvando e lendo conteúdos nesta memória a partir de arquivos salvos em disco. Permite, ainda, que *applets* sejam instalados em uma imagem de memória EEPROM. A troca de dados (I/O) é realizada via *sockets* TCP o que permite simular a interação com uma leitora de cartões (CAD) através da implementação dos protocolos T=CL (sem contato) e T=0 e T=1 (com contato) e comandos APDUs, conforme definidos nas ISOs 7816-3 e 7816-4.

A versão do *cref* disponível no *Java Card Toolkit 2.2.2* é uma implementação em 32 bits. Desta forma, ela permite que as imagens de memórias (simulação de *chips Java Card*) ultrapassem o limite de 64 KB disponíveis na versão anterior (SUN MICROSYSTEMS, 2005b, pág. 73).

5.6 Procedimentos Executados

A implementação e teste do protótipo objeto do presente trabalho esteve baseada nas ferramentas e ambientes descritos na seção anterior. Assim, nesta seção são apresentados os procedimentos executados para a estruturação e codificação do referido protótipo.

5.6.1 *Applet Java Card*

Inicialmente, o *applet Java Card* utilizado no protótipo foi desenvolvido e testado por meio da IDE Eclipse e do *plugin* IBM JCOP. Posteriormente, este *applet* foi carregado,

instalado e também testado nos cartões JCOP adquiridos para o projeto, conforme previa a metodologia inicial aplicada ao desenvolvimento do protótipo. Entretanto, devido às limitações, dificuldades e incompatibilidades tecnológicas entre os *chips* adquiridos e os celulares testados, descritas na seção 5.4.4, esta metodologia teve de ser descontinuada e substituída por métodos que utilizam simulação de *smart cards* e emulação de telefones celulares.

Assim, a metodologia do projeto passou a considerar a simulação do ambiente e a utilização de um *applet Java Card* de propriedade da Sun Microsystems, distribuído junto ao *Java Card Toolkit* sob o nome de *Wallet.java*. Esse *applet* encontra-se instalado em um arquivo de imagem de *smart card* do tipo C-Language *Java Card* RE – *cref* denominado *demo2.eeprom*, também de propriedade da Sun Microsystems e distribuído junto ao produto *Sun Java Wireless Toolkit 2.5.2*. O *applet Wallet.java* apresenta uma estrutura de funcionamento descrita a seguir.

5.6.1.1 Declaração de variáveis estáticas de controle

As variáveis de controle possuem os códigos hexadecimais que devem ser recebidos e transmitidos através dos comandos APDUs. O trecho de código fonte a seguir apresenta essas variáveis:

```
public class Carteira extends Applet {

    // Código do byte CLA byte do cabeçalho do comando APDU
     final static byte CARTEIRA_CLA  = (byte)0x80;

    // Códigos do byte INS do cabeçalho do comando APDU
     final static byte VERIFICA      = (byte) 0x20;
     final static byte CREDITO       = (byte) 0x30;
     final static byte DEBITO        = (byte) 0x40;
     final static byte GET_SALDO     = (byte) 0x50;


    // Saldo máximo
     final static short SALDO_MAX                        = 0x7FF
    // Total máximo por transação
     final static byte TOTAL_MAX_POR_TRANSACAO           = 127;
    // Número máximo de tentativas de autenticação do PIN antes de bloqueá-lo
     final static byte LIMITE_TENTATIVAS_PIN             =(byte)0x03;
    // Tamanho máximo do PIN
     final static byte TAMANHO_MAX_PIN                   =(byte)0x08;
    // Falha na verificação do PIN
     final static short SW_FALHA_NA_VERIFICACAO          = 0x6300;
```

```
    // PIN é requerido para transações de débito e credito
     final static short SW_VERIFICACAO_DO_PIN_REQUERIDA  = 0x6301;
    // Valor de transação inválido
     final static short SW_TOTAL_POR_TRANSACAO_INVALIDO  = 0x6A83;
    // Saldo excedido caso a transação seja realizada
     final static short SW_SALDO_MAXIMO_EXCEDIDO         = 0x6A84;
    // Saldo negativo caso a transação seja realizada
     final static short SW_SALDO_NEGATIVO                = 0x6A85;

    ...
}
```

5.6.1.2 Métodos transacionais do *applet Java Card*

Durante a execução de uma transação um objeto APDU é recepcionado pelo cartão e processado através do método `process (APDU apdu)`. O objeto APDU carrega consigo um *array* de *bytes* (*buffer*) utilizado para transferir o cabeçalho do comando e dados entre o dispositivo CAD e o *smart card*. Neste momento somente os cinco primeiros *bytes* [`CLA, INS, p1, p2, Lc`] estão disponíveis no buffer APDU. A seguir é transcrito o código fotnte do método `process (APDU apdu)`.

```
public void process(APDU apdu) {
    // Aloca o buffer APDU em um array de bytes para leitura
     byte[] buffer = apdu.getBuffer();
     ...

    // Verifica se o byte CLA possui um código que especifica um comando válido na estrutura do applet
     if (buffer[ISO7816.OFFSET_CLA] != CARTEIRA_CLA)
         ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    // Verifica o conteúdo do byte INS e direciona para a transação do applet
    // Caso o código INS seja válido, caso contrário retorna o código do erro correspondente
     switch (buffer[ISO7816.OFFSET_INS]) {
         case GET_SALDO:
             getSaldo(apdu);
             return;
         case DEBITO:
             debit(apdu);
             return;
         case CREDITO:
             credit(apdu);
             return;
         case VERIFICA:
             verify(apdu);
             return;
         default:
             ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
     }
}
```

Conforme demonstrado no trecho de código fonte acima, as transações são selecionadas a partir do conteúdo do *byte* INS. Caso esse código não corresponda a uma transação válida pré-definida (`0x20, 0x30, 0x40 ou 0x50`) o cartão retorna o erro correspondente (`ISO7816.SW_INS_NOT_SUPPORTED`) ao dispositivo CAD.

Supondo-se que o byte INS contenha o código `0x30`, o fluxo é então direcionado para a uma transação de crédito. Essa transação é muito semelhante à transação de débito e contém a seguinte estrutura:

```
private void credit(APDU apdu) {

    // Verifica se o PIN foi validado
      if ( ! pin.isValidated() )
          ISOException.throwIt(SW_VERIFICACAO_DO_PIN_REQUERIDA);
    // Aloca o buffer APDU em um array de bytes para leitura.
      byte[] buffer = apdu.getBuffer();

    // O byte Lc informa o número de bytes do campo data field do comando APDU.
      byte numBytes = buffer[ISO7816.OFFSET_LC];

    // Lê e armazena o número de bytes constantes após o quinto byte do cabeçalho APDU.
      byte byteRead = (byte)(apdu.setIncomingAndReceive());

    // Verifica se o número de bytes lidos corresponde com o número informado no campo Lc.
      if (( numBytes != 1 ) || (byteRead != 1))
          ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    // Lê e armazena o valor do crédito.
      byte valorDoCredito = buffer[ISO7816.OFFSET_CDATA];

    // Verifica se o valor do crédito satisfaz os requisitos de valor por transação.
      if ((valorDoCredito > TOTAL_MAX_POR_TRANSACAO) || (valorDoCredito < 0))
          ISOException.throwIt(SW_TOTAL_POR_TRANSACAO_INVALIDO);

    // Verifica se o novo saldo satisfará o requisito de saldo máximo.
      if ( (short)( saldo + valorDoCredito)  > SALDO_MAX )
          ISOException.throwIt(SW_SALDO_MAXIMO_EXCEDIDO);

    // Realiza o crédito na variável saldo.
      saldo = (short)(saldo + valorDoCredito);

}
```

De outra forma, supondo-se que o byte INS contenha o código `0x50`, o fluxo é então direcionado para a uma transação de consulta de saldo. Essa transação apresenta a seguinte estrutura:

```
private void getSaldo(APDU apdu) {
    // Aloca o buffer APDU em um array de bytes para leitura.
      byte[] buffer = apdu.getBuffer();

    // Informa que o applet finalizou o processamento e que o sistema deve preparar a construção de uma
    // resposta APDU contendo um data field.
      short le = apdu.setOutgoing();
      if ( le < 2 )
          ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    // Informa ao CAD o número de bytes que serão retornados.
      apdu.setOutgoingLength((byte)2);

    // Move os dados do saldo no buffer APDU para o deslocamento 0 do data field.
      buffer[0] = (byte)(saldo >> 8);
      buffer[1] = (byte)(saldo & 0xFF);

    // Envia 2 bytes referentes ao saldo a partir do deslocamento 0 do data field do buffer APDU.
      apdu.sendBytes((short)0, (short)2);
}
```

Ainda, supondo que o byte INS contenha o código `0x20`, o fluxo é então direcionado para a transação de verificação e validação do PIN. Essa transação apresenta a seguinte estrutura:

```
private void verifica(APDU apdu) {

    // Aloca o buffer APDU em um array de bytes para leitura.
      byte[] buffer = apdu.getBuffer();

    // Lê e armazena a informação do PIN para validação.
      byte byteRead = (byte)(apdu.setIncomingAndReceive());

    // Verifica o PIN lido no buffer através do campo ISO7816.OFFSET_CDATA
      if ( pin.check(buffer, ISO7816.OFFSET_CDATA, byteRead) == false )
          ISOException.throwIt(SW_FALHA_NA_VERIFICACAO);
}
```

5.6.1.3 Imagem de memória EEPROM e *cref*

Conforme citado anteriormente, o *applet Java Card* é executado através do simulador *C-Language Java Card RE – cref* e do arquivo de imagem distribuído pela Sun Microsystems juntamente com o *Sun Java Wireless Toolkit 2.5.2*. O *cref* é inicializado através da console de comando do sistema operacional e para tanto devem ser informados alguns parâmetros que indicam a porta em que será estabelecido o *socket* TCP, além dos arquivos de entrada e saída.

Tendo em vista que o protótipo compreende a execução de dois dispositivos móveis, cada um com um *smart card*, devem ser executadas duas instâncias do simulador *cref*, uma para cada cartão simulado, observando que diferentes instâncias devem estabelecer *sockets* em diferentes portas TCP.

As execuções das instâncias do *cref* devem ser realizadas através das seguintes linhas de comando:

```
C:\cref –z –p 9025 –i demo2.eeprom –o demo2.eeprom
C:\cref –z –p 9026 –i demo2.eeprom –o demo2.eeprom
```

Por padrão estabelecido pelo *Sun Java* WKT 2.5.2, para a simulação são utilizadas as portas TCP 9025 e 9026. A figura 5.9 ilustra a execução de uma instância do simulador *cref*.

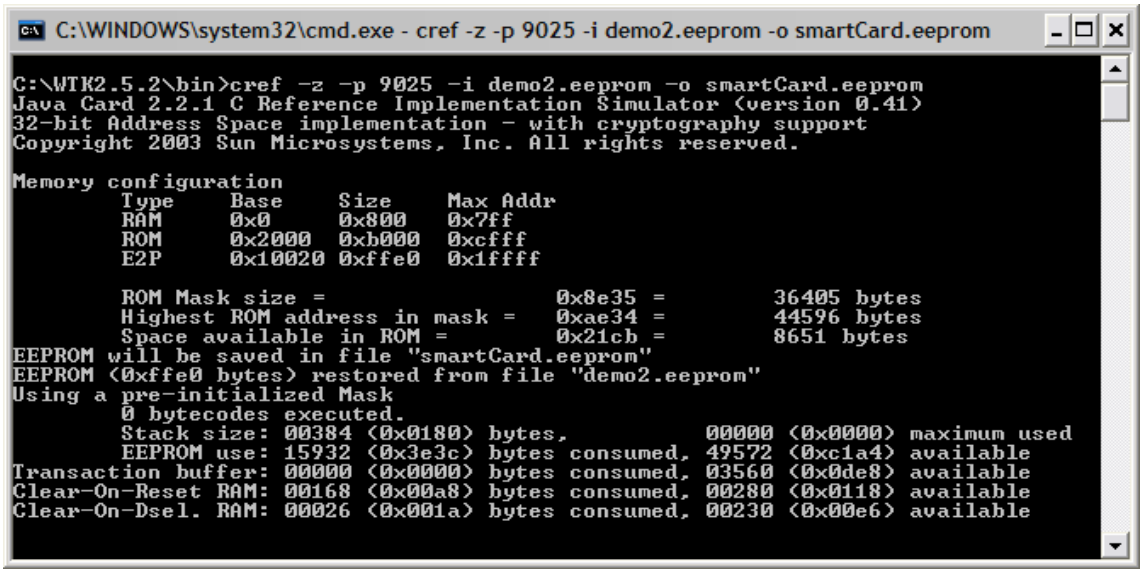

Figura 5.9 – Execução do *C-Language Java Card RE – cref*.
Fonte: Elaborada pelo autor.

5.6.2 Codificação do MIDlet *host*

A codificação do MIDlet *host* foi dividida em duas partes principais, separadas e distintas pela sua atuação. Inicialmente foram desenvolvidas as classes e os métodos responsáveis pela apresentação, comunicação e troca de mensagens APDUs com os dispositivos *Java Card*. Paralelamente foram implementadas a classe e os métodos responsáveis pela comunicação e troca de mensagens *Bluetooth* entre os dois dispositivos móveis. Após os testes e validações pertinentes, em um segundo momento, os dois MIDlets foram, então, integrados. Como resultado obteve-se o MIDlet denominado *CarteiraBluetoothMIDlet*, contendo o protótipo alvo do presente trabalho.

5.6.2.1 CarteiraBluetoothMIDlet

O MIDlet Carteira*Bluetooth*MIDlet está baseado em três classes denominadas *CarteiraBluetoothMIDlet.java*, *FuncoesCarteiraBluetooth.java*, *Converter.java*, inseridas no pacote *PackageTCC*.

A classe principal, Carteira*Bluetooth*MIDlet.java implementa o código responsável pelo estabelecimento da comunicação com o *smart card*, criação do Servidor e Cliente *Bluetooth*, tratamento das mensagens da comunicação *Bluetooth*, criação dos formulários e menus, bem como os métodos `CommandListener`, `CommunicationListener`, `ConnectionListener`, `ItemCommandListener`, além dos métodos obrigatórios para o controle do MIDlet J2ME. A seguir são demonstrados trechos do código fonte responsáveis por estabelecer a comunicação com o *smart card*. Os trechos referentes a comunicação *Bluetooth* são detalhados na seção 5.6.2.2 – *Framework* Marge.

```
// Bibliotecas
import javax.microedition.apdu.APDUConnection;
import javax.microedition.io.Connector;

public class Carteira Bluetooth MIDlet extends MIDlet implements CommandListener,
CommunicationListener, ConnectionListener, ItemCommandListener {

    // Variável que armazena o número do slot e o AID do applet alvo da conexão.
      private final String Slot0 = "apdu:0;target=a0.00.00.00.62.03.01.0c.06.01";

    // Variável que armazena a conexão
      public static APDUConnection carteiraConnection;


    // Abre e fecha uma conexão com o smart card no slot e com o applet informados.
          try {
             carteiraConnection = (APDUConnection) Connector.open( Slot0 );
```

```
            ...
            // Espaço para a troca de comandos APDUs com o smart card
            ...

            carteiraConnection.close();
         } catch (Exception e) {
            try {
               carteiraConnection.close();
            } catch (Throwable t) { }
         }
}
```

A classe *FuncoesCarteiraBluetooth.java* implementa as funções responsáveis por ler os valores de crédito, débito e PIN, realizar as trocas de comandos APDUs com o *smart card*, receber e tratar as mensagens de resposta recebidas, além de manter uma matriz de *bytes* com a estrutura dos comandos APDUs de crédito, débito, consulta de saldo e autenticação do PIN. A seguir são demonstrados trechos do código fonte responsáveis por trocar comandos APDUs com o *smart card*.

```
public class FuncoesCarteiraBluetooth {

    // Códigos para identificação de commandos APDU na matriz
      private static final int AUTENTICA_PIN  = 0;
      private static final int CONSULTA_SALDO = 1;
      private static final int CREDITO        = 2;
      private static final int DEBITO         = 3;

      static byte[] tempAPDU;
      static byte[] resposta;


    // Matriz com estrutura dos commandos APDUs
      public static final byte [][] carteiraBluetoothAPDUs = {

          // APDU de validação do  PIN
          {
            (byte)0x80, (byte)0x20, (byte)0x00, (byte)0x00, (byte)0x05,
            (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x7F
          },

          // APDU de consulta saldo
          {
            (byte)0x80, (byte)0x50, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x02
          },

          // APDU de crédito
          {
            (byte)0x80, (byte)0x30, (byte)0x00, (byte)0x00, (byte)0x01, (byte)0x00,
            (byte)0x7F
          },

          // APDU de débito
          {
            (byte)0x80, (byte)0x40, (byte)0x00, (byte)0x00, (byte)0x01, (byte)0x00,
            (byte)0x7F
          },
      };

      ...
```

```
    public static String creditaValor( String valor ){

        try {
          // Converte o valor do crédito para um array de bytes.
            byte[] aux = Converter.int2ByteArray(Integer.parseInt(valor));


          // Utiliza a estrutura do comando APDU de crédito, alterando o quarto byte com o valor do crédito
            tempAPDU = carteiraBluetoothAPDUs[CREDITO];
            tempAPDU[5] = aux[3];

          // Envia um APDU para processamento no smart card e recebe a resposta do applet Java Card.
            resposta = CarteiraConnection.exchangeAPDU(tempAPDU);

          // Converte a resposta de hexadecimal para string e a retorna ao código que a chamou.
            return trataResposta( Converter.toHexString(resposta));

        } catch (IOException ex) {
             ex.printStackTrace();
             return "ERRO";
        }
    }
    ...
}
```

Por sua vez, a classe *Converter.java* implementa as funções responsáveis pelas conversões entre valores decimais, hexadecimais e *strings*, necessários para a interação com o *smart card*.

5.6.2.2 *Framework* Marge

Marge é um *framework* para desenvolvimento de aplicações Java ME e Java SE criado por Bruno Cavaler Ghisi e Lucas Torri, da Universidade Federal de Santa Catarina. Seu objetivo principal é facilitar o uso da API JSR 82 – *Java APIs for Bluetooth*, uma vez que essa API se mostra um tanto complexa para desenvolvedores iniciantes na tecnologia *Bluetooth*.

O Marge trabalha como uma camada de abstração de métodos nativos da API JSR 82, tais como conexão, configuração de protocolos, troca de mensagens, informações, pesquisa e busca por dispositivos. A figura 5.10 ilustra a localização do *framework* Marge nas camadas que compõe uma aplicação Java ME (https://marge.dev.java.net/).

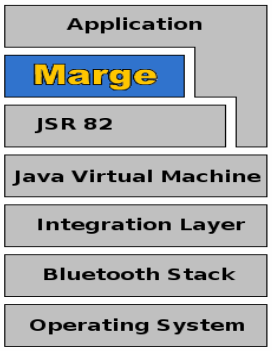

Figura 5.10 – Localização do *framework* Marge nas camadas de uma aplicação Java.
Fonte: https://marge.dev.java.net/

Implementação padrão

A implementação padrão oferece recursos de busca por dispositivos e serviços, além de uma gama de outros recursos. Basicamente é necessário iniciar um servidor e então um cliente que irá pesquisar por dispositivos *Bluetooth* ativos (servidor), buscar por um serviço específico em um dispositivo e então conectar-se a ele. Após esses procedimentos ambos estarão habilitados para trocar mensagens. Para o correto funcionamento dos processos de busca e troca de mensagens é necessária a implementação de algumas interfaces, a saber:

a) ***CommunicationListener*** – é utilizada para que os dispositivos servidor e cliente sejam notificados sobre o recebimento de mensagens e possíveis erros no recebimento;

b) ***ConnectionListener*** – quando uma conexão é estabelecida o dispositivo servidor é notificado e retorna mensagens através desta interface;

c) ***InquiryListener*** – é a interface utilizada para retornar os dispositivos encontrados;

d) ***ServiceSearchListener*** – é a interface utilizada para retornar os serviços encontrados.

AutoConnect

Uma maneira simplificada de estabelecer uma comunicação *Bluetooth* entre dois dispositivos móveis é utilizar os métodos *AutoConnect.createServer* e *AutoConnect.connectToServer* presentes na biblioteca *net.java.dev.marge.autocon. AutoConnect*. Também exige que sejam implementadas as interfaces *CommunicationListener* e *ConnectionListener*.

Entretanto, o *AutoConnect* não permite implementar a busca por dispositivos e serviços. Assim, deve-se ter o cuidado de verificar se o ambiente em que se executará a aplicação não possui dispositivos *Bluetooth* indesejáveis, pois o cliente poderá se conectar a outro dispositivo que esteja executando o mesmo serviço.

Tendo em vista as restrições de tempo encontradas durante a execução deste projeto optou-se pela implementação utilizando os métodos *AutoConnect.* A seguir são demonstrados trechos do código fonte implementados para estabelecer e utilizar uma conexão *Bluetooth* para troca de mensagens:

a) **Criando o dispositivo Servidor:**

```
// Bibliotecas
import net.java.dev.marge.autocon.AutoConnect;
import net.java.dev.marge.communication.CommunicationListener;
import net.java.dev.marge.communication.ConnectionListener;
import net.java.dev.marge.entity.Device;
import net.java.dev.marge.entity.ServerDevice;

public class CarteiraBluetoothMIDlet extends MIDlet implements CommandListener,
CommunicationListener, ConnectionListener, ItemCommandListener {

     private Device device;

    // Criando um Server através da classe AutoConnect.
     AutoConnect.createServer( SERVER_NAME, CommunicationListener(),
     ConnectionListener() );

    // Conexão estabelecida
     public void connectionEstablished( ServerDevice serverDevice, RemoteDevice
     remoteDevice ) {
         this.device = serverDevice;
         this.connected = true;
     }
}
```

b) **Criando a conexão do dispositivo Cliente**

```
     private Device device;

    // Estabelecendo uma conexão com o Server e atribuindo-a a um dispositivo.
     CarteiraBluetoothMIDlet.this.device = AutoConnect. connectToServer(SERVER_NAME,
     CommunicationListener());
```

c) **Trocando mensagens entre os dispositivos conectados**

```
     device.send("Mensagem a ser enviada!!!".getBytes());
```

d) **Recebendo mensagens – Servidor e Cliente**

```
     public void receiveMessage( byte[] msgRecebida){
         String s = new String (mensagemRecebida);
     };
```

e) **Tratando erros**

```
public void errorOnReceiving(IOException e) {
    itemProgresso.setText(e.toString());
}
public void errorOnSending(IOException e) {
    itemProgresso.setText(e.toString());
}
public void errorOnConnection(IOException e) {
    itemProgresso.setText(e.toString());
}
```

Para maiores detalhes sobre o funcionamento e documentação do *framework* Marge recomenda-se a consulta ao *web site* do projeto no endereço eletrônico *https://marge.dev.java.net/* e às aplicações de demonstração que lá estão disponíveis.

5.6.3 Comunicações durante uma transação

A realização de uma transação envolve a troca de mensagens entre os dispositivos com o objetivo de controlar a interação entre os telefones celulares até a concretização das transações de crédito e débito. Ainda, como já mencionado, existem trocas de comandos APDUs entre a aplicação *host* e o *smart card*. Essas mensagens são exploradas a seguir.

5.6.3.1 Fluxo de mensagens Bluetooth e APDUs

Após ser inicializado, um dispositivo servidor fica aguardando a conexão de um dispositivo cliente, para, então, estar apto a realizar uma transação de transferência de dinheiro eletrônica entre o par da aplicação. Uma vez que os dois dispositivos estão pareados, o dispositivo cliente solicita que o seu usuário insira o PIN e o valor da transação para iniciar o procedimento de débito. O sistema verifica se o valor pretendido atende aos requisitos de valor máximo por transação e saldo do cliente. Caso os critérios sejam satisfeitos, envia uma mensagem ao dispositivo servidor contendo o valor da transação e fica aguardando uma resposta positiva do mesmo para prosseguir com a transação de débito.

Ao receber o valor de uma transação de crédito, o dispositivo servidor verifica se o valor satisfaz o requisito de valor máximo por transação e então solicita que o usuário informe o PIN que dará acesso a transação de crédito. Uma vez validado o PIN o servidor envia uma confirmação da autenticação para o dispositivo cliente que retomará o seu fluxo, concretizando o débito do valor solicitado. Tão logo o débito é realizado, uma mensagem é

enviada ao dispositivo servidor para que o mesmo retome o seu fluxo e finalize a transação de crédito.

As mensagens enviadas e recebidas pelos dispositivos durante as transações de débito e crédito são constituídas de códigos textuais que informam o estado daquela transação dentro do fluxo normal do processo e iniciam uma nova etapa do fluxo no dispositivo pareado. A seqüência de trocas de mensagens é ilustrada na figura 5.11:

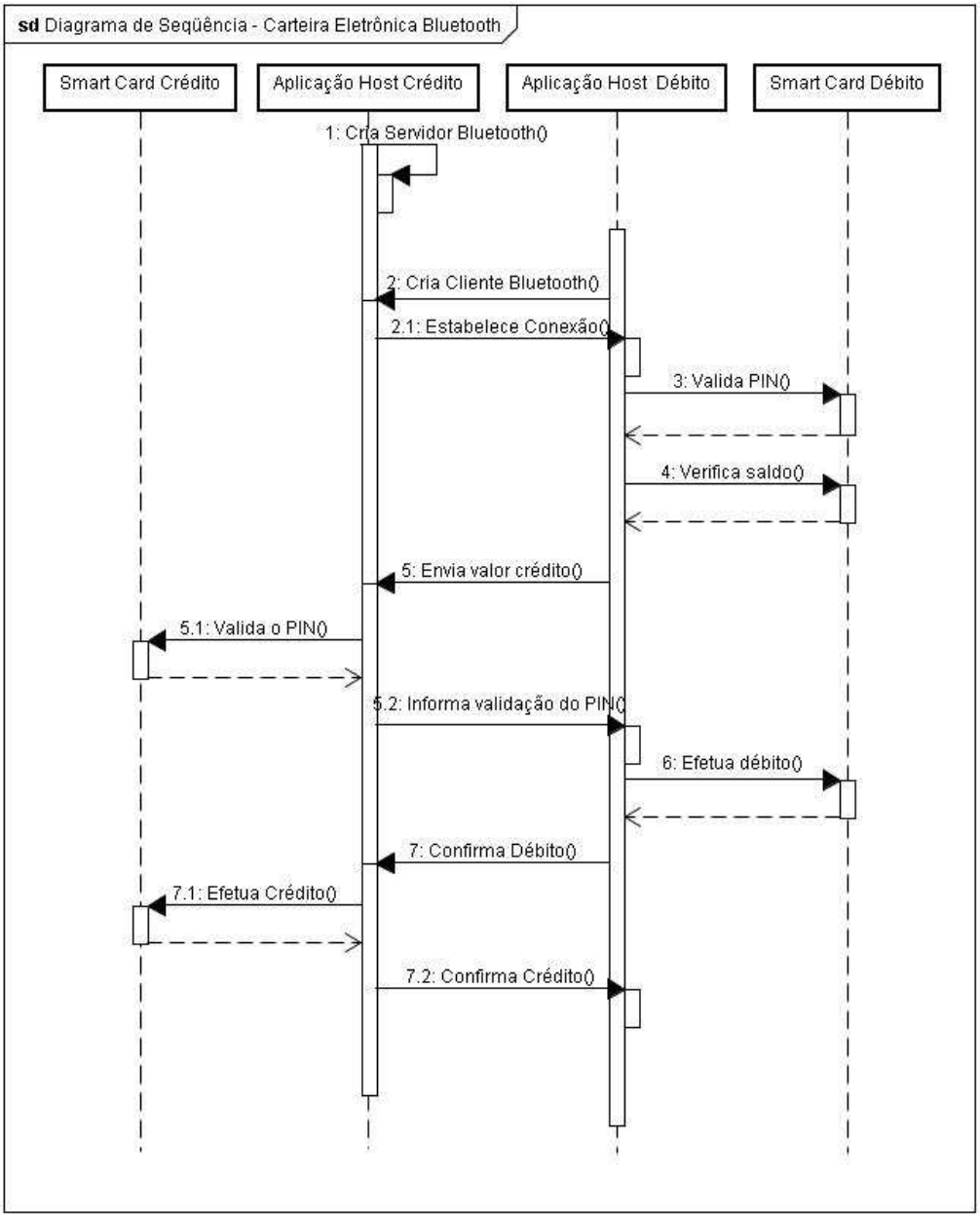

Figura 5.11 – Diagrama de seqüência de mensagens *Bluetooth* e comandos APDUs.
Fonte: Elaborada pelo autor.

5.6.3.2 Análise de comandos APDUs

O *Sun Java Wireless Toolkit 2.5.2* possui um recurso denominado *Network Monitor* que é capaz de registrar o fluxo dos comandos APDUs trocados entre a aplicação *host* e o *applet Java Card*. Desta forma é possível visualizar todos os campos de um APDU e o seu respectivo conteúdo. As figuras 5.12 a 5.17 ilustram o fluxo de comandos APDUs em um dispositivo Servidor que estabelece uma conexão com um *smart card*, valida um PIN e realiza uma transação de crédito. Pode-se observar que para cada comando monitorado é possível visualizar o seu respectivo conteúdo, segmentando e descrevendo as informações transitadas em cada campo do protocolo.

A figura 5.12 ilustra o comando para criação de uma conexão com a aplicação de AID a0.00.00.00.62.03.01.0c.06.01 no slot zero. A figura 5.13 apresenta o comando APDU de resposta do *applet Java Card* que retorna o código 90 00, enviados através dos campos SW1 e SW2, informando que a operação foi realizada com sucesso.

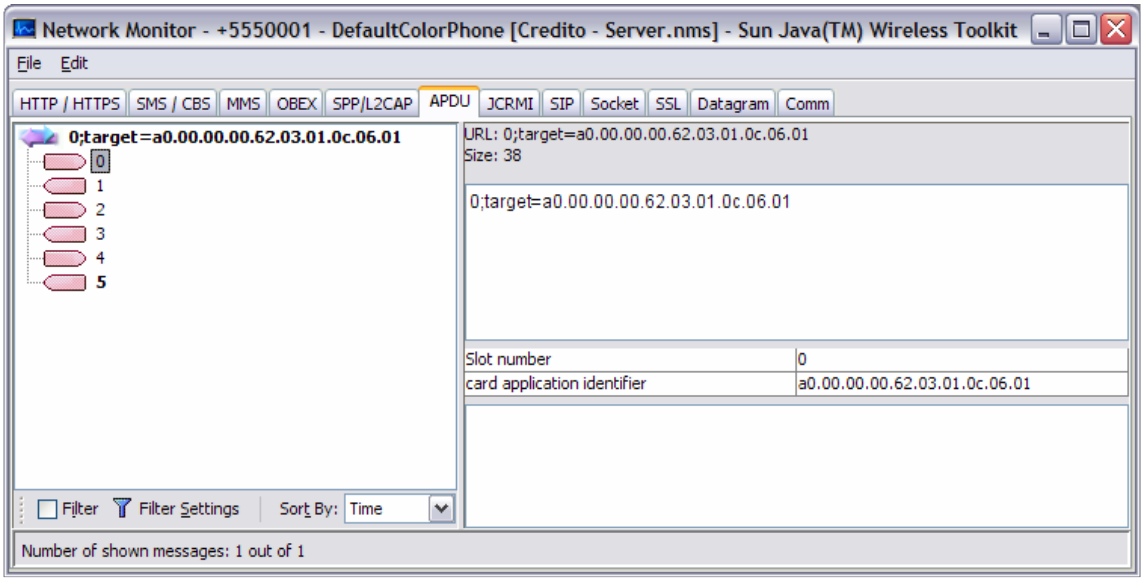

Figura 5.12 – Estabelecendo a conexão com o *smart card*.
Fonte: Elaborada pelo autor.

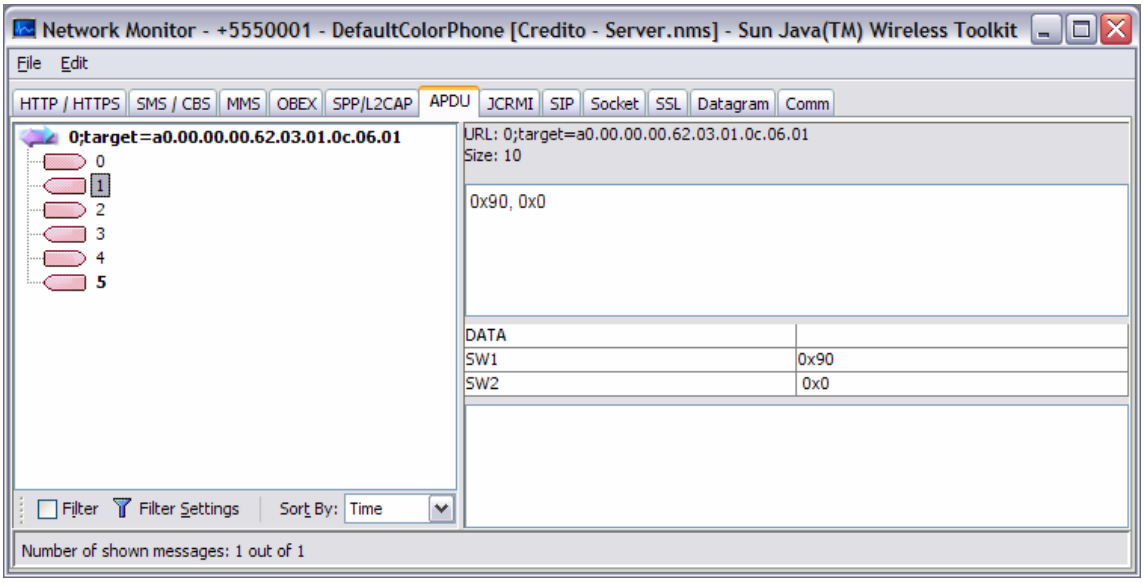

Figura 5.13 – Resposta do *applet* para a solicitação de conexão.
Fonte: Elaborada pelo autor.

A figura 5.14 apresenta o envio de um comando APDU de solicitação de validação do PIN da aplicação *host* para o *applet Java Card*. A figura 5.15 ilustra o comando APDU de resposta do *applet*, informando, através do código 90 00 constantes nos campos SW1 e SW2, que o PIN foi autenticado com sucesso.

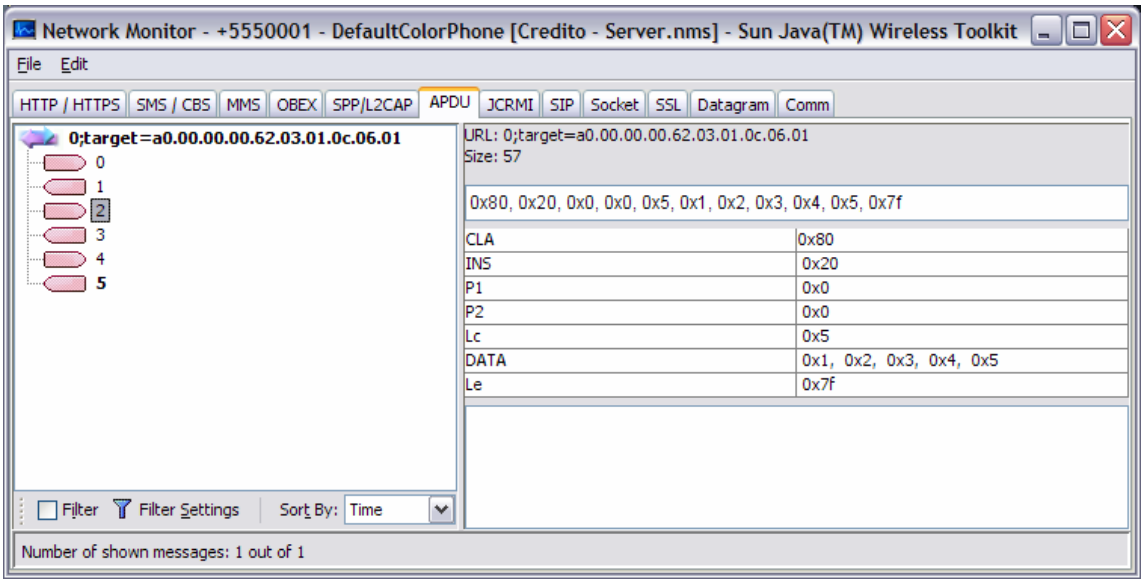

Figura 5.14 – APDU para validação do PIN.
Fonte: Elaborada pelo autor.

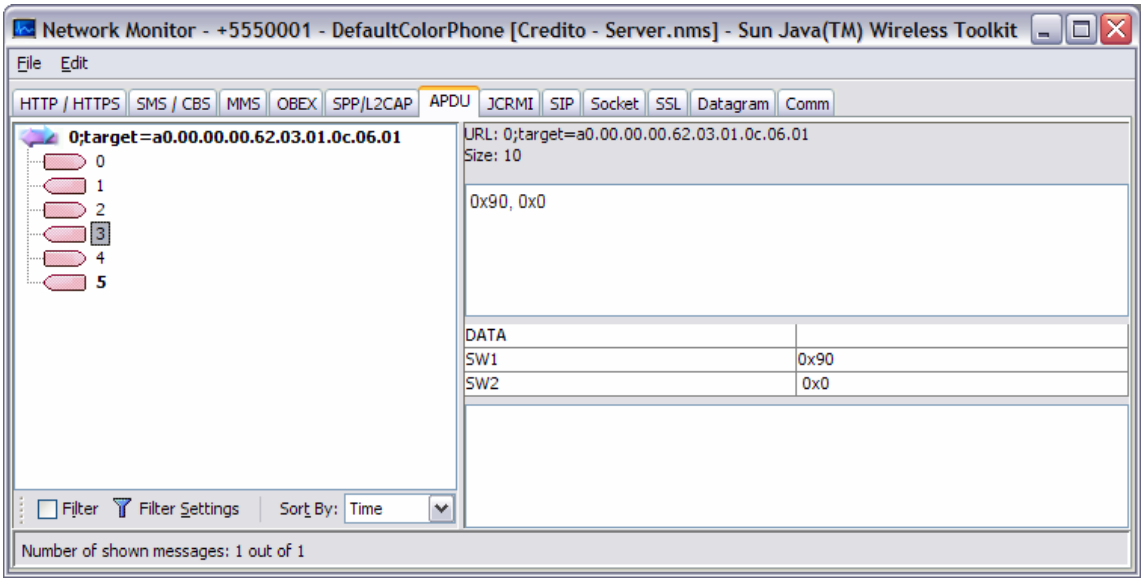

Figura 5.15 – Resposta do *applet* para a solicitação de validação do PIN.
Fonte: Elaborada pelo autor.

A figura 5.16 ilustra o envio de um comando APDU de crédito no valor de $ 45,00, representado pelo valor 0x2d no campo DATA do comando APDU, da aplicação *host* para o *applet Java Card*. A figura 5.17 apresenta o comando APDU de resposta do *applet*, informando, através do código 90 00 constantes nos campos SW1 e SW2, que a transação foi realizada com sucesso.

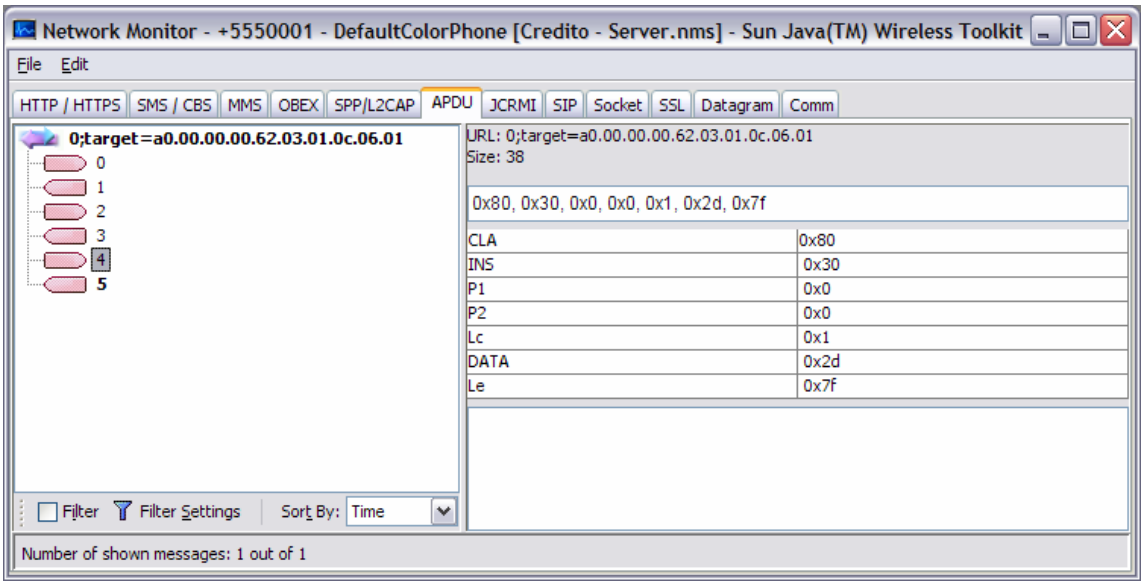

Figura 5.16 – APDU de transação de crédito.
Fonte: Elaborada pelo autor.

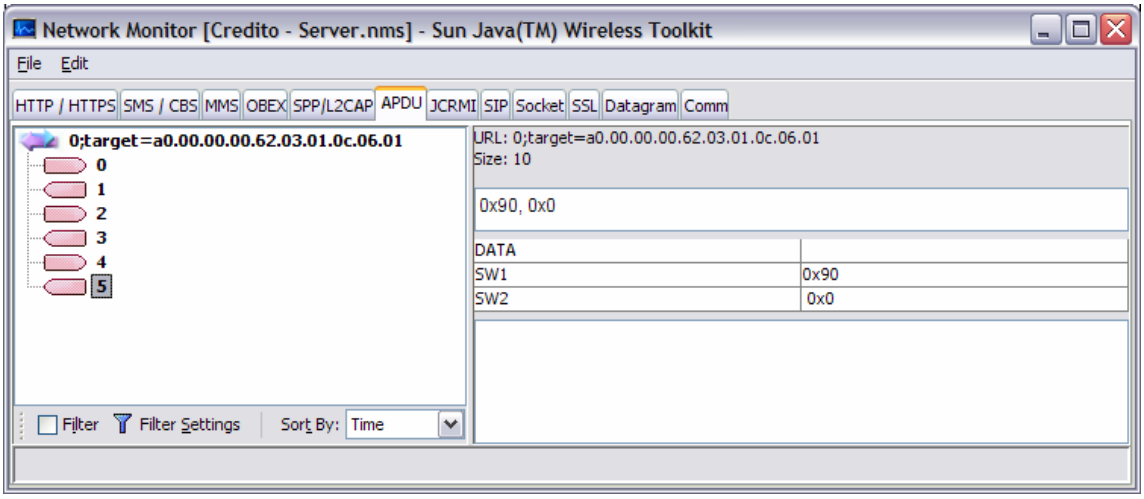

Figura 5.17 – Resposta do *applet* para o comando de crédito.
Fonte: Elaborada pelo autor.

Cabe citar que o fluxo de comandos APDUs para as transações em um dispositivo Cliente ocorrem de forma bastante semelhante a apresentada acima. Ainda, os códigos acima apresentados podem ser visualizados nos trechos de código fonte apresentados nas seções 5.6.1 e 5.6.2.

6 RESULTADOS

Neste capítulo são apresentados os resultados obtidos e observados após o desenvolvimento do protótipo, objeto principal do presente trabalho. Assim, a seguir é descrito o funcionamento das quatro funções disponibilizadas, juntamente com exemplos de transações e ilustrações do comportamento do aplicativo, bem como são abordadas as limitações verificadas no protótipo, propostas de trabalhos futuros e implicações acadêmicas e gerenciais.

Cabe salientar que para todas as funções descritas a seguir é imprescindível que o simulador *C-Language Java Card RE – cref* esteja sendo executado e tenha como entrada a imagem de um *smart card*, *Java Card*, que contenha o *applet Wallet.java*, disponibilizado pela *Sun Microsystems*.

6.1 Função Autentica PIN

A função autentica PIN tem como objetivo autorizar o acesso às funções de Crédito e Débito antes do início do fluxo das mesmas. Seu atributo de entrada é uma senha numérica composta por cinco dígitos, informada pelo usuário em campo específico na área superior do visor do dispositivo móvel. Antes de ser enviada para validação no *applet*, a aplicação *host* verifica se o PIN informado satisfaz o requisito de cinco dígitos. Caso esse critério seja satisfeito a mesma é enviada para validação no *smart card* por meio da função *FuncoesCarteiraBluetooth.autenticaPIN()*.

A aplicação *host* foi desenvolvida para que uma autenticação do PIN seja válida para uma única transação de crédito ou débito. Esse controle é realizado através da criação e encerramento de conexões, também chamadas de sessões, com o *smart card*, de forma que a cada nova conexão é necessária uma nova autenticação do PIN.

Dentre os comportamentos possíveis estão PIN autenticado com sucesso, tamanho de PIN diferente de cinco dígitos e PIN incorreto. O *applet* utilizado neste protótipo, disponibilizado pela *Sun Microsystems*, está programado com o PIN `12345`. Esse programa controla o número de autenticações incorretas, de forma que caso o PIN seja erroneamente informado por três vezes consecutivas a conexão com o *smart card* é encerrada. A figura 6.1 ilustra o comportamento do protótipo quando solicitada a função autentica PIN.

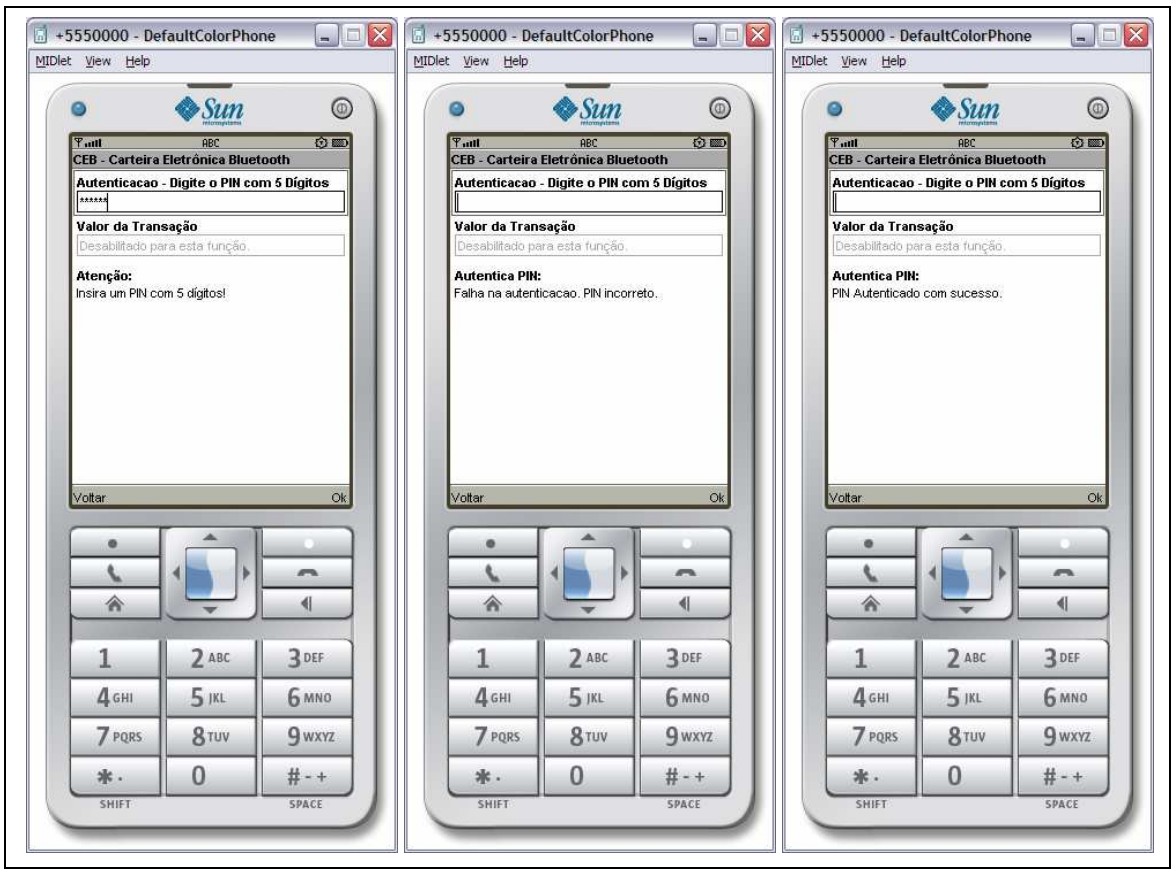

Figura 6.1 – Função autentica PIN.
Fonte: Elaborada pelo autor.

6.2 Função Consulta Saldo

A função consulta saldo tem o propósito de informar ao usuário o valor monetário armazenado na carteira eletrônica e está disponível por meio da opção *Consulta Saldo* do menu principal. Esse menu faz uma chamada à função

FuncoesCarteiraBluetooth.consultaSaldo() e tem como resposta o saldo armazenado na variável *saldo* do *applet Java Card*. É uma transação local que não necessita da autenticação do PIN para ser executada, nem faz uso das chamadas para conexões *Bluetooth*.

Como já citado, é imprescindível que o simulador *cref* esteja sendo executado com um *applet* válido para a aplicação *host*. A figura 6.2 ilustra a seqüência de eventos para a utilização da função Consulta Saldo. No exemplo ilustrado, observa-se a tela inicial da função, seguida pela solicitação de confirmação de uso do *smart card* e, então, pela resposta recebida do *applet Java Card*, que informa um saldo de R$ 50,00 na carteira eletrônica.

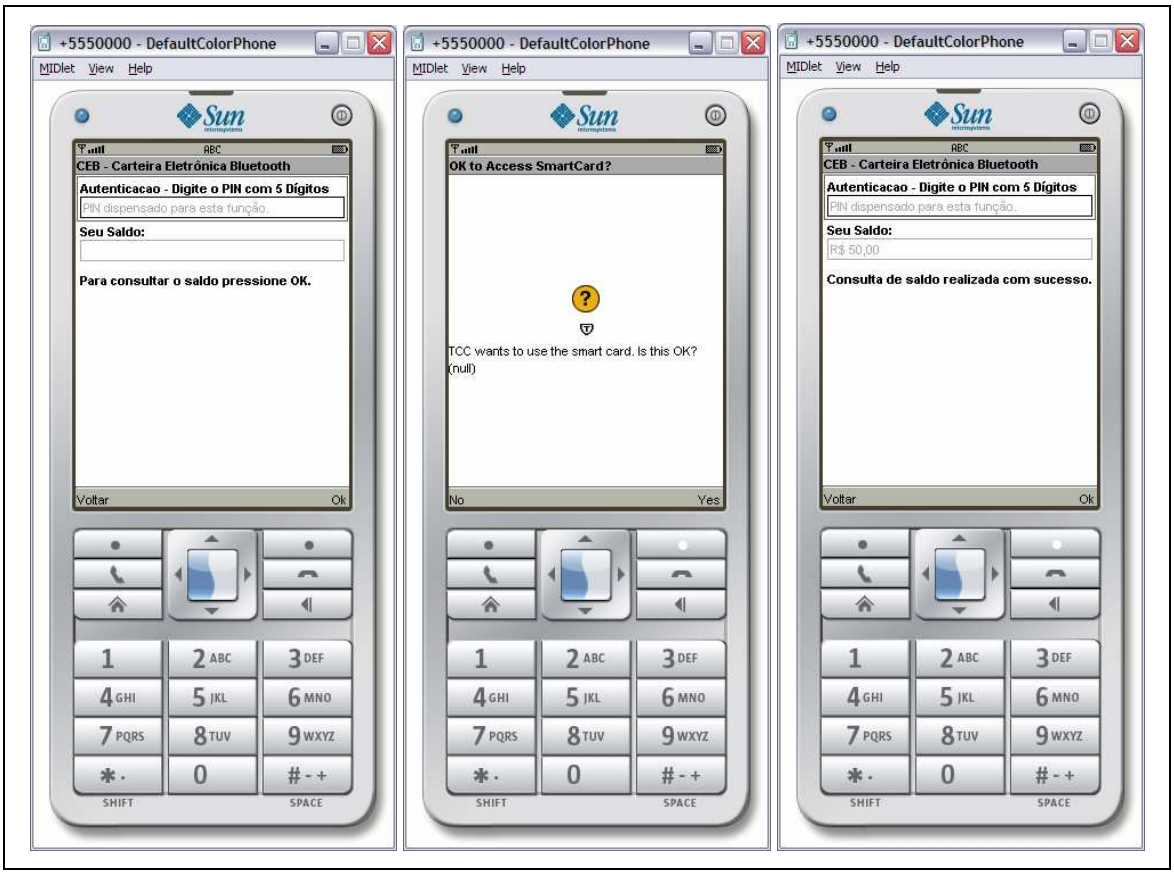

Figura 6.2 – Função Consulta Saldo.
Fonte: Elaborada pelo autor.

Observa-se que a solicitação de confirmação para uso do *smart card* é realizada somente uma vez para cada execução da aplicação *host* Carteira Eletrônica *Bluetooth*. É importante citar que não se trata de uma confirmação ou requisito inserido durante o desenvolvimento, mas, sim, de procedimentos da plataforma das APIs utilizadas.

6.3 Função Crédito

A função Crédito tem como objetivo principal aceitar um valor informado por outro dispositivo com o qual o dispositivo está pareado, somando-o ao saldo armazenado na variável *saldo* do *applet Java Card*. Adicionalmente, a função Crédito possui outro objetivo que tange a criação de um servidor de conexões *Bluetooth* e o estabelecimento de uma conexão com um dispositivo cliente que iniciará uma transação para transferência de valores monetários. Da mesma forma que a função Débito, a função Crédito não é capaz de ser concluída sem a interação com um dispositivo que execute a função oposta.

O acesso à função Crédito é realizado através do menu principal da aplicação *host*. Por convenção estabelecida durante o desenvolvimento do protótipo, um dispositivo que executa essa função e que receberá créditos sempre será o dispositivo Servidor de conexões *Bluetooth*. Desta forma, uma vez que se estabeleceu um acordo para transferência de valores entre dois dispositivos, o aparelho receptor do crédito será o responsável por iniciar a seqüência de procedimentos para a transferência de fundos. Para o usuário isso ocorrerá de forma transparente, pois a criação do servidor *Bluetooth* ocorre de forma automática no momento em que é escolhida a opção *Crédito (Server)* do menu principal da aplicação *host*. Tão logo é criado o servidor *Bluetooth*, a aplicação fica aguardando por conexões vindas de dispositivos clientes, ou seja, que estarão debitando valores em seu *smart card* e enviando-os ao dispositivo servidor. A figura 6.3 ilustra a inicialização da função Crédito descrita anteriormente.

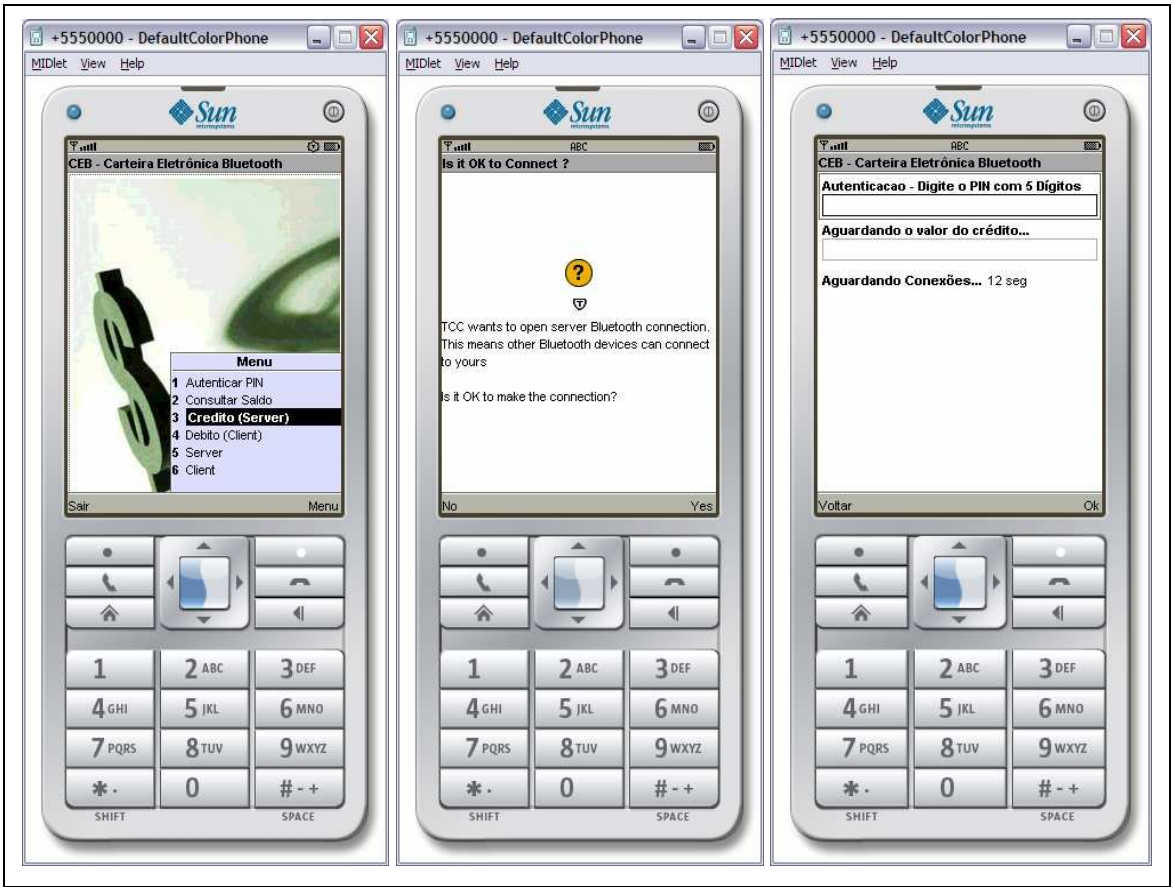

Figura 6.3 – Função Crédito.
Fonte: Elaborada pelo autor.

O dispositivo Servidor fica, então, aguardando por conexões oriundas de dispositivos Clientes que virão a estabelecer uma comunicação através da utilização da função Débito, no menu inicial da sua aplicação de carteira eletrônica. Após estabelecida uma conexão *Bluetooth* com um dispositivo cliente, o Servidor entra em estado de espera por um valor a ser creditado. Assim que o Cliente envia um valor para crédito de fundos o Servidor identifica e recepciona esse valor, solicitando em seguida que o usuário insira o seu PIN para aceitá-lo. A figura 6.4 ilustra os passos descritos neste parágrafo. Observa-se que o Servidor está recebendo um valor de R$ 34,00 enviados pelo dispositivo Cliente e que deve inserir o seu PIN para aceitar o crédito recebido.

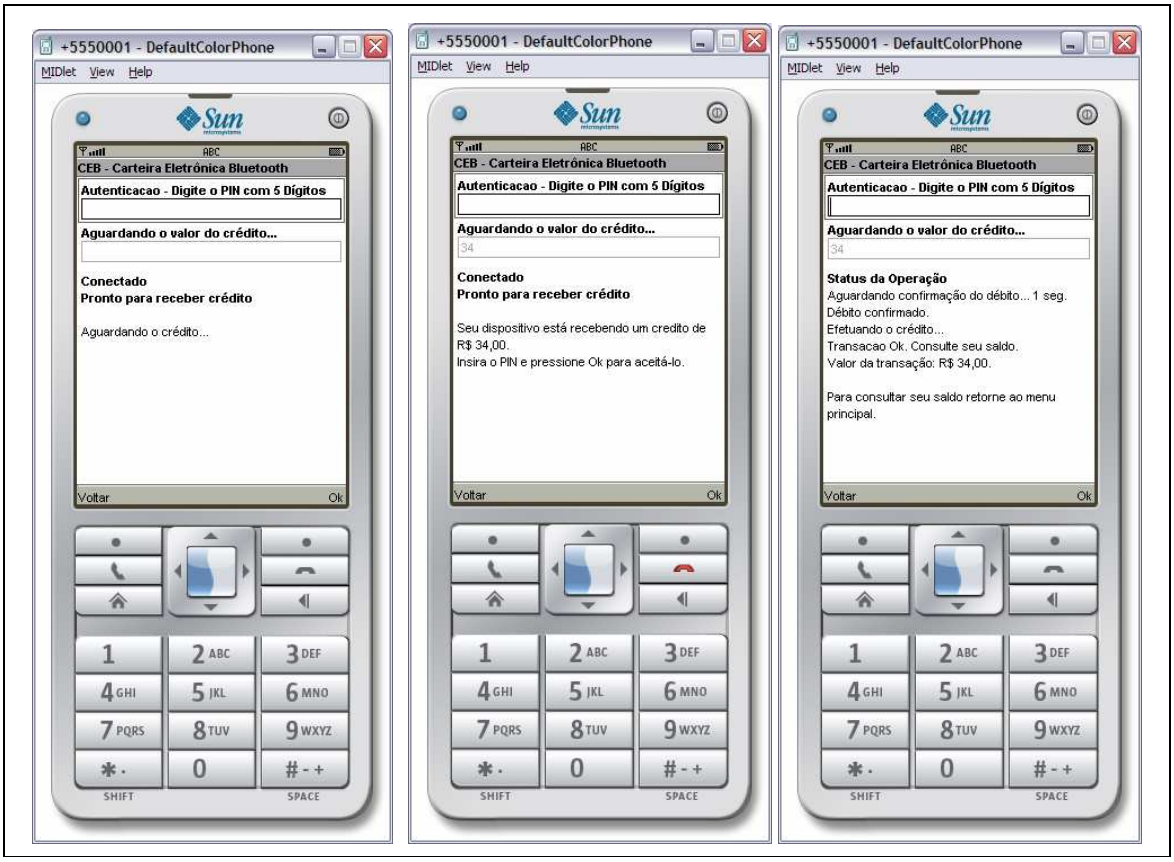

Figura 6.4 –Finalização da função Crédito.
Fonte: Elaborada pelo autor.

Assim que o dispositivo Servidor efetua a validação do PIN informado pelo usuário é enviada uma mensagem ao dispositivo Cliente e o seu fluxo é interrompido para que o Cliente concretize o débito e retorne a confirmação do mesmo. Tão logo o Servidor receba a confirmação do débito pelo Cliente, o mesmo retoma o seu fluxo e concretiza o crédito do valor recebido, enviando uma mensagem de confirmação do crédito ao dispositivo Cliente e, então, finalizando a operação de transferência de fundos.

6.4 Função Débito

A função Débito tem como objetivo principal enviar valores monetários a outro dispositivo que execute a aplicação Carteira Eletrônica *Bluetooth*, debitando-os do seu saldo armazenado no *smart card*, mais precisamente na variável *saldo* do *applet Java Card*. Adicionalmente, essa função também tem o objetivo de estabelecer uma conexão *Bluetooth*

com um dispositivo Servidor que já tenha sido criado pela função Crédito e esteja aguardando por conexões.

O acesso a função Débito deve ser realizado através do menu principal da aplicação Carteira Eletrônica *Bluetooth*, na opção *Débito (Client)*. Por convenção estabelecida durante o desenvolvimento da aplicação *host* Carteira Eletrônica *Bluetooth*, o dispositivo que executa a função Débito será o dispositivo Cliente na conexão *Bluetooth* a ser estabelecida. Ainda, da mesma forma que a função Crédito, a função Débito não é capaz de ser concluída sem a interação com um dispositivo que execute a função oposta.

Assim, uma vez estabelecido um acordo de transferência de fundos entre dois usuários e já inicializado o dispositivo Servidor, cabe ao usuário Cliente acessar a função Débito para estabelecer a conexão *Bluetooth* e iniciar o fluxo da transferência de fundos. Inicialmente é solicitada ao usuário uma confirmação para a criação de um cliente de conexão *Bluetooth* e, em seguida, é solicitado que sejam informados o PIN e o valor da transação. A aplicação *host* verificará se o PIN atende os requisitos de tamanho e se o valor informado está dentro da faixa de valores permitida para uma transação, além de verificar se o valor solicitado é coberto pelo saldo existente na carteira eletrônica. Por questões de segurança, as verificações do saldo e do valor máximo permitido por transação são novamente realizadas pelo *applet* no *smart card*. A figura 6.5 ilustra o fluxo inicial de uma transação de débito.

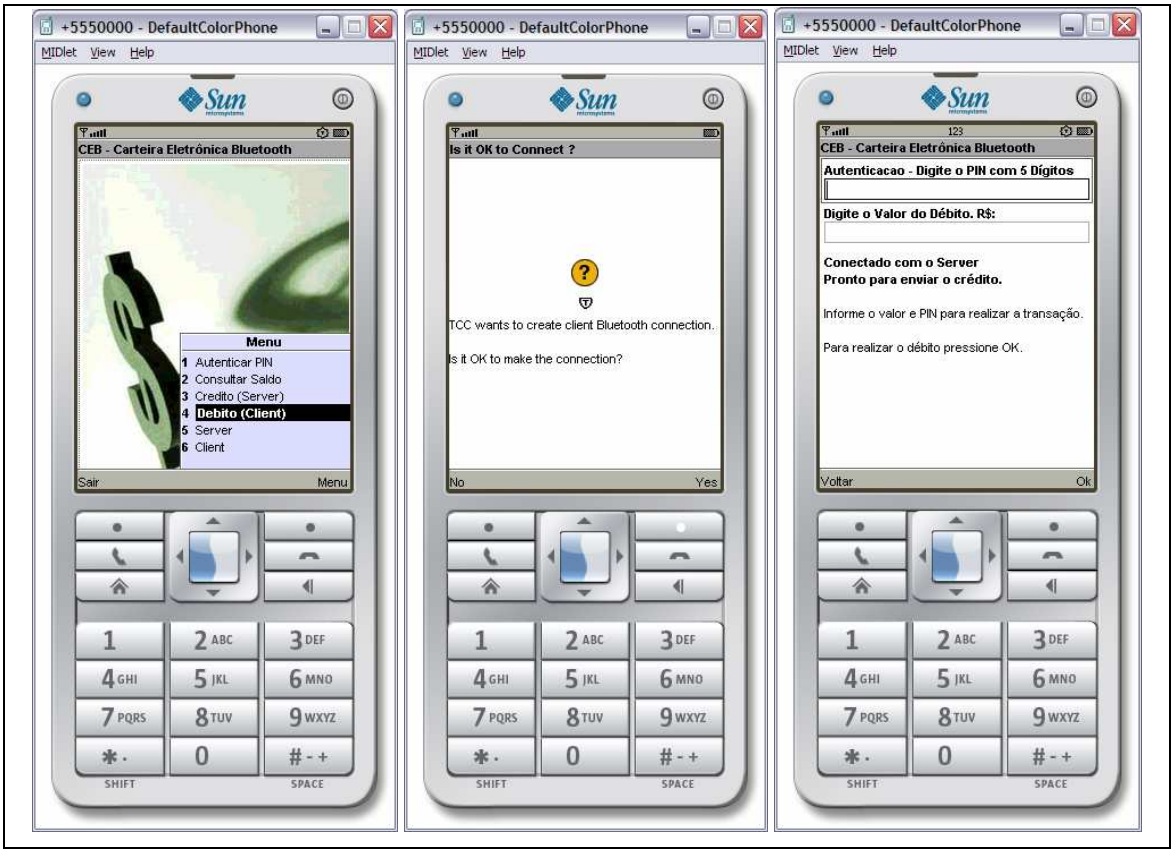

Figura 6.5 – Função Débito.
Fonte: Elaborada pelo autor.

Tão logo O PIN seja validado e o valor solicitado autorizado, o dispositivo Cliente envia uma mensagem ao dispositivo Servidor informando o valor da transação e paralisando o seu fluxo enquanto aguarda pela confirmação do PIN no dispositivo que será realizado o crédito. Assim que Cliente recebe a confirmação de que o PIN para o crédito foi autenticado pelo Servidor, retoma o seu fluxo, efetiva o débito e envia uma mensagem de confirmação do débito ao Servidor que efetivará o crédito. Ainda, ficará aguardando a confirmação do crédito pelo Servidor para então finalizar a transação de transferência de fundos. A figura 6.6 ilustra o fluxo final de telas e mensagens no dispositivo cliente para uma transação de teste de R$ 34,00.

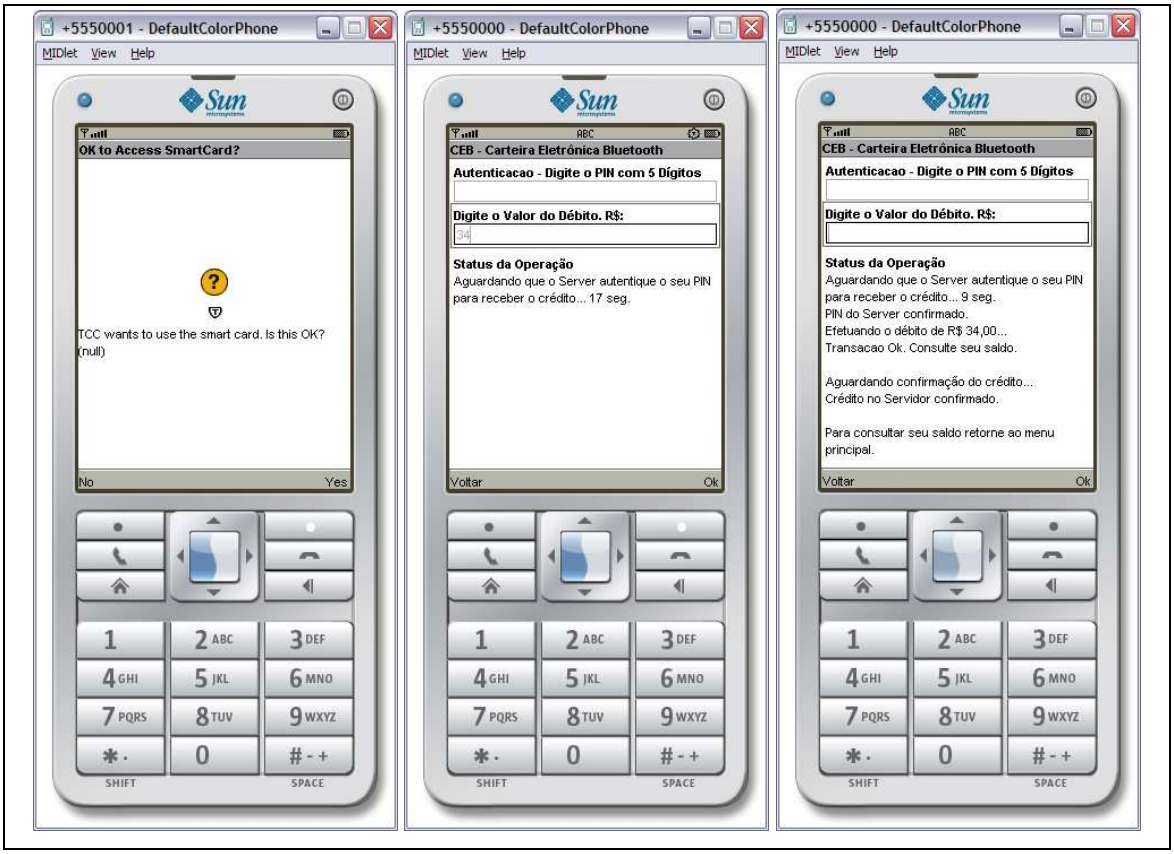

Figura 6.6 – Finalização da função Débito.
Fonte: Elaborada pelo autor.

As verificações de saldo e valor limite por transação podem gerar exceções que são tratadas e apresentadas ao usuário. Tendo em vista que o modelo é voltado para transações de pequenos e médios montantes, foi convencionado que as transações devem respeitar um valor mínimo de R$ 1,00 e máximo de R$ 100,00, bem como o saldo máximo armazenado na carteira eletrônica não pode ser maior que R$ 32.000,00. Outra exceção possível e tratada diz respeito a impossibilidade da aplicação *host* estabelecer uma conexão com o *smart card*, e conseqüentemente não acessar o *applet Java Card*. Essa exceção aplica-se não somente a função débito, mas a todas as funções já descritas anteriormente.

A figura 6.7 ilustra as exceções aqui relatadas, através da tentativa de transferência de valor de R$ 51,00 enquanto o saldo da carteira eletrônica é de R$ 50,00, tentativa de transferência de valor de R$ 102,00 e tentativa mal sucedida de conexão com o *smart card* devido a não inicialização do simulador *cref.*

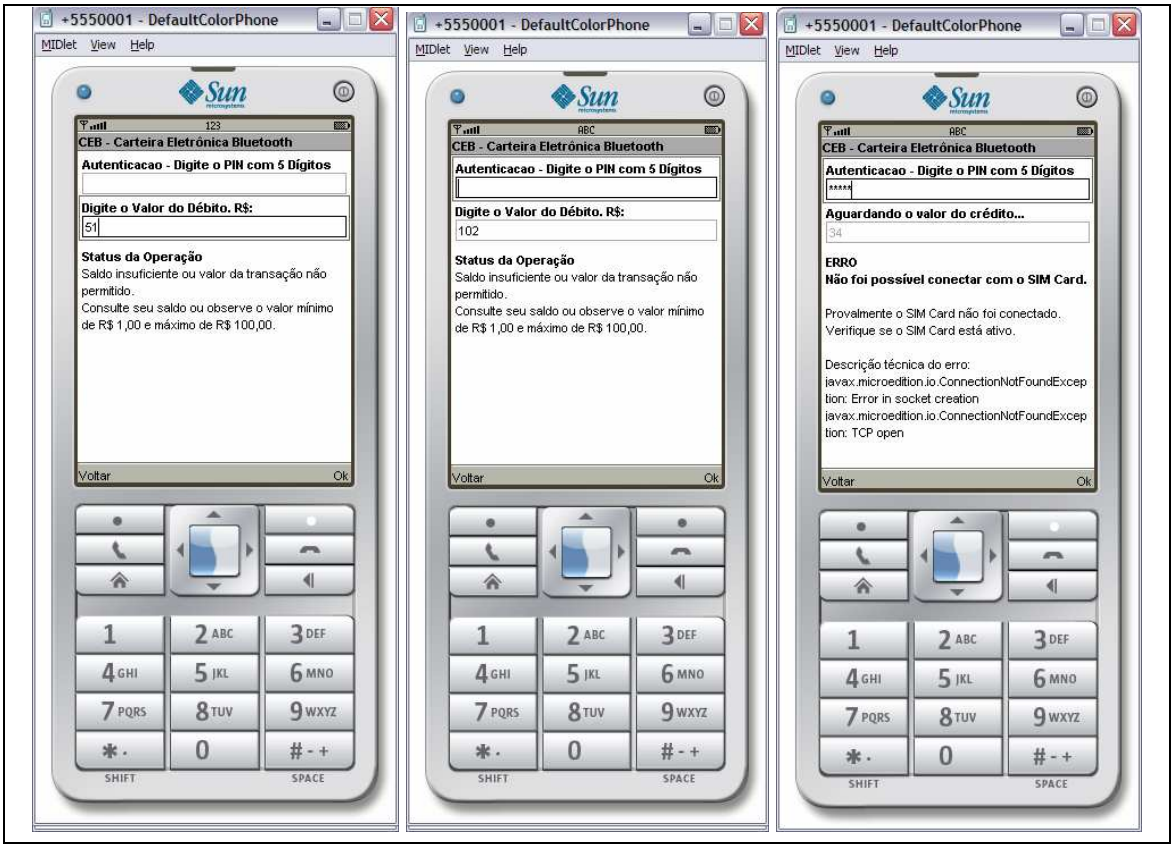

Figura 6.7 – Exceções tratadas no fluxo das transações da carteira eletrônica.
Fonte: Elaborada pelo autor.

6.5 Limitações

Após o desenvolvimento deste projeto, podem ser identificadas algumas limitações que devem ser consideradas em trabalhos futuros. Dentre essas, está o fato do protótipo não ter sido validado junto a telefones celulares reais, o que possibilitaria uma melhor demonstração e avaliação do modelo proposto. Outras limitações identificadas dizem respeito a não implementação de funções mais complexas de controle do fluxo de mensagens entre os dispositivos que compõem o par de uma transação, funções de controle da efetividade das operações de débito e crédito, controle e opção de estorno de transações que encontrem algum problema durante o seu fluxo, funções avançadas de criptografia e segurança da comunicação, além de um módulo voltado para a instituição administradora do sistema. Por fim, entende-se que outra limitação está associada a não consideração de outras tecnologias de comunicação

wireless entre telefones celulares, como, por exemplo, o estabelecimento de uma rede *wi-fi* entre esses dispositivos.

Cabe salientar que as limitações apontadas não invalidam a pesquisa, uma vez que o objetivo deste projeto é a prova de conceitos que permeiam um sistema de carteira eletrônica e não a construção de um sistema de pagamento eletrônico completo.

6.6 Trabalhos Futuros

Sistemas de pagamento eletrônico, por sua natureza, são sistemas complexos que devem considerar controles eficientes para garantir a integridade, confidencialidade e disponibilidade das suas informações. Essa complexidade deve ser explorada profundamente na busca por soluções cada vez mais seguras e eficientes. Neste sentido, cabe, mais uma vez, salientar que esta pesquisa não se esgota nesse trabalho. Muitas são as questões que devem ser abordadas em trabalhos futuros e algumas delas são brevemente comentadas a seguir:

a) **Busca por dispositivos e serviços e seleção dos mesmos** – a implementação da aplicação *host* utilizou somente o método *AutoConnect* do *framework* Marge para a busca e conexão a dispositivos e serviços. É desejável que a aplicação localize os aparelhos com sinal *Bluetooth* ativo na área e selecione aquele com o qual irá parear. O *framework* Marge disponibiliza os métodos para que esta função seja implementada. Ainda, devem ser estudadas maneiras de garantir a segurança da comunicação através dos recursos da especificação *Bluetooth*;

b) **Utilização de certificados digitais, infra-estrutura de chaves públicas e chaves criptográficas de sessão**: é imprescindível que recursos tecnológicos de hardware ou software garantam a confidencialidade das informações trafegadas por meio da conexão *Bluetooth*. Assim, a utilização de certificados digitais e uma infra-estrutura de chaves públicas apresenta-se como uma importante, e talvez a mais importante, linha de pesquisa a ser seguida e aprofundada. Conforme já citado, chaves criptográficas privadas podem ser armazenadas de forma segura nos chips SIM *cards*, juntamente com a sua parte pública e o respectivo certificado digital. A troca de chaves públicas e certificados permitiria a validação da identidade de um dispositivo e a cifragem das mensagens trocadas de maneira que somente o receptor

pudesse decifrá-las. Como apresentado no capítulo 2, os SIM *cards* possuem recursos de hardware suficientes para a realização de cálculos criptográficos que envolvem a cifragem e decifragem de informações. Ainda, modelos criptográficos para a criação de chaves de sessão baseadas em critérios pseudo-randômicos podem ser aplicados para garantir que cada transação ou sessão seja cifrada por uma chave criptográfica simétrica diferente, integralmente calculada e criada dentro do *smart card*;

c) **Desenvolvimento do módulo administrador**: dando continuidade ao desenvolvimento do projeto e com o objetivo de finalizar a arquitetura do modelo proposto, faz-se necessária a implementação do módulo administrador do sistema, responsável por fazer a troca de dinheiro eletrônico por moeda física, e vice-versa, além do desbloqueio e alteração do PIN da carteira eletrônica;

d) **Avaliação criteriosa dos aspectos nativos de segurança dos *smart cards*, dos softwares desenvolvidos e da tecnologia *Bluetooth* implementada**: após o desenvolvimento e implantação das camadas de segurança é desejável que sejam direcionados esforços de pesquisa para validação dos recursos de segurança implementados em busca de fragilidades e vulnerabilidades, seja em nível de transporte, armazenamento ou processamento de informações;

e) **Implementação de funções de controle complexas**: é desejável que o sistema considere funções complexas de controle do fluxo de mensagens entre a aplicação *host* dos dispositivos envolvidos, de forma a garantir a efetividade das operações nos dois lados da transação. Também, convém que sejam implementadas funções de estorno de transações que encontrem algum problema durante o seu fluxo;

f) **Validação do protótipo em dispositivos físicos**: por fim, outra proposta para futuros trabalhos permeia a aquisição de SIM *cards* cujo AID do *card manager* seja conhecido, ou seja, que possibilite ao desenvolvedor a instalação de *applets* em sua memória e, conseqüentemente, a validação do protótipo em dispositivos físicos.

Também, a busca por telefones que aceitem e interajam com os cartões JCOP adquiridos pode ser outro caminho a ser seguido. Ainda, sabe-se que a troca de informações entre uma aplicação J2ME e um SIM *card* real esconde questões complexas de implementação e que exigem esforço de pesquisa para serem resolvidas.

Os temas anteriormente propostos compõem uma lista de necessidades que, num primeiro momento, apresentam-se como prioridades a serem trabalhadas. Certamente surgirão necessidades adicionais que, da mesma forma, deverão ser foco de pesquisas futuras e ajudarão a tornar o modelo proposto mais robusto e confiável.

6.7 Implicações Acadêmicas e Gerenciais

Esse trabalho, ao longo do seu desenvolvimento, foi voltado para casos de uso em que os recursos armazenados e trocados entre os dispositivos pertencem ao Sistema Financeiro Nacional. Entretanto, é importante salientar que esse modelo pode ser implementado em diferentes âmbitos, bastando que sejam definidos critérios e regras rígidas para o controle dos recursos transferidos entre os dispositivos móveis. Deve-se ter em mente que os valores armazenados não precisam necessariamente fazer referência à moeda do Sistema Financeiro Nacional. Basta que a instituição administradora do sistema defina e acorde com seus usuários a espécie de moeda a que o sistema está atrelado.

Desta forma, o modelo proposto pode ser amplamente utilizado no âmbito interno de uma grande empresa que disponibiliza créditos para serem trocados nos seus restaurantes, cafeterias, setor de cópias ou algum outro setor de prestação de serviços internos. Para isso, basta que um acordo seja firmado entre todas as partes usuárias do modelo. A implementação desse modelo, muito provavelmente, traria automação e segurança a processos de pagamentos de créditos diversos, uma vez que existe uma redução do fluxo de dinheiro físico em caixas, além de uma possível redução de custos.

Ainda, entende-se que o presente trabalho traz contribuições científicas ao meio acadêmico, seja por iniciar um projeto que pode ser aprofundado e continuado, ou por estudar tecnologias ainda pouco acessíveis e exploradas em cursos de graduação de áreas de Tecnologia, como *Smart Card, Java Card, Bluetooth* e desenvolvimento para dispositivos

móveis. Também, espera-se que esta pesquisa traga à comunidade acadêmica uma importante discussão sobre uma promissora espécie de pagamento eletrônico em dispositivos móveis, como as carteiras eletrônicas.

CONCLUSÃO

Os sistemas de pagamento eletrônico têm entrado em uma nova era, a era das tecnologias e dispositivos móveis. O avanço e a disseminação das redes de comunicações *wireless*, aliados à evolução do poder computacional e dos elementos de segurança dos telefones celulares e PDAs, como funções criptográficas fortes, protocolos e hardwares seguros, juntamente com a possibilidade de interação de aplicações J2ME com *applets* residentes em SIM *cards*, passaram a permitir que desenvolvedores de softwares criem soluções de pagamento eletrônico seguras a esses aparelhos. Somada a esses fatores está a popularização desses dispositivos, o que faz com que a mobilidade seja apontada por especialistas como uma grande tendência de arquitetura para serviços de pagamento eletrônico.

De todas as tecnologias de segurança que surgiram nos últimos anos, a tecnologia *Smart Card* foi, sem dúvida, aquela que conseguiu possibilitar a construção de sistemas de pagamento eletrônico com alto grau de segurança. Foi somente após o advento dessa tecnologia que se tornaram possíveis e viáveis os sistemas de carteira eletrônica.

A tecnologia *Smart Card* teve um grande avanço no momento em que seus sistemas operacionais passaram a suportar ambientes multi-aplicações. A partir da incorporação de uma máquina virtual Java e do *Java Card Runtime Environment* foi possível adicionar valor à tecnologia *Smart Card* através da capacidade de suporte de diferentes aplicações em um único cartão. Ainda, essa tecnologia se mostra altamente padronizada e amparada por padrões técnicos estabelecidos por consórcios de empresas e órgãos mundialmente reconhecidos como líderes do setor.

O presente trabalho teve como objetivo estudar os conceitos acerca de um sistema de pagamento eletrônico e as tecnologias envolvidas, bem como a implementação de um

protótipo deste tipo de pagamento baseado em *smart cards*, para transações ponto-a-ponto em dispositivos móveis. O desenvolvimento do protótipo demonstrou que é possível a criação de aplicações que possibilitem o pagamento eletrônico por meio de transferência de fundos armazenados de forma segura em *smart cards*. Demonstrou, também, que é possível que esses valores sejam transmitidos através de uma comunicação *Bluetooth*, ponto a ponto, entre dois aparelhos celulares simulados padrão GSM, sem o envolvimento da rede de telefonia celular e/ou um servidor de aplicação para validação das transações.

Os resultados obtidos foram satisfatórios e demonstraram que, dentro do escopo abordado, é possível e viável a construção de soluções de pagamento eletrônico para transações *off-line* em dispositivo móveis. A implementação da API JSR-177 *Security & Trust* por parte do ambiente de simulação *Sun Java Wireless Toolkit 2.5.2* permitiu a construção de uma aplicação *host* capaz de estabelecer uma comunicação com um *smart card* simulado e trocar informações com o mesmo através de comandos APDUs. Teoricamente a mesma solução simulada é aplicada a dispositivos móveis reais cuja JVM implemente a API JSR-177. Também, a implementação da API JSR 82 pela mesma ferramenta, em conjunto com os métodos disponibilizados pelo *framework* Marge, permitiu a criação do ambiente necessário para a simulação da conexão *Bluetooth* entre os dois aparelhos celulares simulados, possibilitando a troca de valores armazenados nos *smart cards* simulados por meio de mensagens *Bluetooth*.

A arquitetura segura dos *smart cards* permite que informações críticas sejam armazenadas de forma segura e inviolável no seu interior, enquanto a plataforma *Java Card*, utilizando-se de uma linguagem de alto nível, disponibiliza uma ferramenta e um ambiente para a construção de aplicativos com alto grau de segurança para *smart cards*. Entretanto, cabe ao futuro do projeto uma análise criteriosa sobre os aspectos de segurança no armazenamento de valores e, principalmente, na transmissão dos mesmos por meio da comunicação *Bluetooth*, avaliando e testando os aspectos de segurança nativos das tecnologias envolvidas e dos softwares desenvolvidos.

A análise de resultados compreendeu, além de transações fictícias, a análise do tráfego de mensagens APDUs geradas entre os dispositivos que compõe o par da aplicação. A transferência de valores simbólicos apresentou um comportamento dentro dos padrões esperados, seguindo o fluxo proposto para uma transação, realizando as funções de débito e crédito de valores corretamente. A análise das mensagens APDUs ilustrou a arquitetura de um

comando APDU e a distribuição dos códigos e mensagens entre seus campos, validando o referencial teórico pesquisado e apresentado.

REFERÊNCIAS BIBLIOGRÁFICAS

BADDELEY, Michelle. Using e-cash in the new economy: an economic analysis of micropayments system. **Journal of Electronic Commerce Research**, Cambridge, vol. 5, nº.4, p.239 – 253, 2004.

BLUETOOTH SIG. The official Bluetooth technology info site. Bluetooth SIG, Inc, 2008. Disponível em <http: //www.bluetooth.com>. Acesso em 27 out. 2008.

BURGE, Mark. Developing Smart card Applications Using the Open*Card* Framework. **ACM Southeast Regional Conference:** Proceedings of the 42nd annual Southeast regional conference. Huntsville, p. 19 – 24, 2004.

CHEN, Zhiqun. **Java Card (tm) Technology for Smart Cards: Architecture and Programmer's (The Java Series) (Paperback)**. Addison-Wesley Professional, 1st edition. 2000. 400p.

EMVCo. Disponível em <http://www.emvco.com>. EMVCo, LLC, 2006. Acesso em 01 de jun. 2008.

FERREIRA, Lucas de Carvalho. **Sistemas de pagamento eletrônico:** classificação, análise e implementação. Campinas: 1999. 128 p. Dissertação (Mestrado em Ciência da Computação) - Instituto de Computação, Unicamp, 1999.

FREITAS, Fábio; LEITE, Tiago. **Bluetooth**. Instituto Superior de Engenharia do Porto. [S.l.], [S.d].

FRODIGH, M; JOHANSSON, P; LARSSON, P. Wireless ad hoc networking - The art of networking without a network. **Ericsson Review**, No. 4, 2002.

GUISI, Bruno. **Marge: *framework* para desenvolvimento de aplicações em Java que façam uso da tecnologia *Bluetooth***. Monografia: 2007. 167 p. (Bacharelado em Sistemas de Informação) – Universidade Federal de Santa Catarina, Florianópolis, 2007.

HONG, Insuk; CHUN, Ingook. The implementation of electronic money for e-commerce using Java Card. **International Symposium on Industrial Electronics**, Pusan, Korea, p. 1369-1372, 2001.

INFODEV, Information for Development Program. **Micro-Payment Systems and Their Application to Mobile Networks.** An infoDev Report. Washington, jan. 2006.

JCOP Tools 3.0 (Eclipse Plugin) – Technical Brief, Revision 1.0. [s.l.], [s.d.], 11 p.

JOHNSON, Thienne M. **Java para dispositivos móveis**: desenvolvendo aplicações com J2ME. São Paulo: Novatec Editora, 2007. 334 p.

KHAN, Faheem. **Securing Java Card applications, Part 1**: Building a Kerberos-enabled J2ME application. [S.l.], 2005.

KNIBERG, Henrik. **What makes a micropayment solution sucessed**. Stockholm: 2002, 68 p. Dissertação (Marter Thesis) - Departament of Applied Information Technology, Kungliga Tekniska Högskolan, 2002.

KOBAYASHI, Carlos Yassunori. **A tecnologia *Bluetooth* e aplicações**. USP, São Paulo, 2004, 5 p.

LI, Peng; ZDANCEWIC, Steve. Advanced Control Flow in Java Card Programming. **LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems**. Washington, vol. 39, p. 165-174, jul. 2004.

LORENZONI, Álvaro. **Smart cards – Java Card**. Monografia: 2006. 134 p. (Bacharelado em Ciência da Computação) - Centro Universitário Feevale, Novo Hamburgo, 2006.

MAHMOUD, Qusay H. **Wireless Application Programming with J2ME and Bluetooth**. [S.l.], 2003a. Disponível em < http://developers.sun.com/mobility/midp/articles/bluetooth1/index.html>. Acesso em : 27 mar. 2008.

_____. **Part II: The Java APIs for Bluetooth Wireless Technology**. [S.l.], 2003b. Disponível em < http://developers.sun.com/mobility/midp/articles/bluetooth2/index.html>. Acesso em: 27 mar. 2008.

MARTINEZ, Antonio Ruiz; MARTINEZ, Daniel Sánchez; MONTESINOS, Maria Martinez; SKARMETA, Antonio F. Gómez. A Survey of Electronic Signature Solutions in Mobile Devices. **Journal of Theoretical and Applied Electronic Commerce Research.** Curicó, Chile, vol. 2, issue 3, p. 94-109, 2007.

MATTOS, Érico Tavares de. **Programação Java para Wireless** / Érico Tavares de Mattos. São Paulo: Digerati Books, 2005. 125 p.

ORTIZ, C. Henrique. **An introduction to Java Card technology:** part 1. [S.l.], 2003a. Disponível em: <http://developers.sun.com/techtopics/mobility/java*card*/articles/java*card*1/>. Acesso em: 27 mar. 2008.

_____. **An introduction to Java Card technology:** part 2, the *Java Card* applet. [S.l], 2003b.Disponível em: <http://developers.sun.com/techtopics/mobility/java*card*/articles/java*card*2/>. Acesso em: 27 mar. 2008.

_____. **An Introduction to Java Card Technology:** Part 3, The *Smart card Host* Application. [S.l.], 2003c.

Disponível em: <http://developers.sun.com/mobility/java*card*/articles/java*card*3/>. Acesso em: 27 mar. 2008.

_____. **Using the Java APIs for Bluetooth Wireless Technology,** Part 1 - API Overview. [S.l], 2004a. Disponível em:

< http://developers.sun.com/mobility/apis/articles/bluetoothintro/index.html />. Acesso em 27 mar. 2008.

_____. **Using the Java APIs for Bluetooth,** Part 2 - Putting the Core APIs to Work. [S.l.], 2004b. Disponível em:

< http://developers.sun.com/mobility/apis/articles/bluetoothcore/index.html >. Acesso em 27 mar. 2008.

_____. **The Security and Trust Services API for J2ME,** Part 1. [S.l.], 2005. Disponível em: <http://developers.sun.com/mobility/apis/articles/satsa1/>.

RANKL, Wolfgang. **Smart card applications:** design models for using and programming *smart cards*. Traduzido por Kenneth Cox. Chichester: John Wiley & Sons Ltd, 2007. 238 p.

RANKL, Wolfgang; EFFING, Wolfgang. **Smart card handbook**. 3 ed. Translated by Kenneth Cox. Chichester: John Wiley & Sons Ltd, 2003. 1088 p.

SCHMEES, Markus. Distributed Digital Commerce. **ICEC '03: Proceedings of the 5th international conference on Electronic commerce**. Oldenburg, p. 131-137, sep. 2003.

SCHREIBER, Flávia; SOUZA, José; GARCIA, Marcelo. **SIM Card**. [S.l]. 2003. Disponível em <http://www.teleco.com.br/tutoriais/tutorialsim/Default.asp>.

SUAVI, Cléber Giovanni. **Documentos e dinheiro eletrônico com smart cards utilizando tecnologia Java Card**. Monografia: 2005. 104 p. (Bacharelado de Ciência da Computação) - Universidade Regional de Blumenau, Blumenau, 2005.

SUN MICROSYSTEMS, Inc. **Runtime Environment Specification**. Java Card™ Plataform, version 2.2.2. Santa Clara: Sun Microsystems, Inc, 2005a, 148 p.

_____. **Development Kit User's Guide**. For the Binary Release with Cryptography Extensions, *Java Card*™ Platform, version 2.2.2. Sun Microsystems, Inc, 2005b, 170 p.

_____. **User's Guide:** Sun Java Wireless Toolkit for CLDC, version 2.5.2. Sun Microsystems, Inc, 2007, 240 p.

_____. **Java APIs for Bluetooth Wireless Technology (JSR 82)**, Specification Version 1.1.1. Sun Microsystems, Inc, 2008, 129 p.

THOMPSON, Timothy J., KLINE, Paul, KUMAR, C. Bala. **Bluetooth® Application Programming With the Java™ APIs Essentials Edition**. Burlington: Elsevier Science & Technology, 2008, 305 p.

WIKI JAVA.NET. **How to start with Marge 0.5**. Java.net the source for Java Technology Collaboration. Disponível em http://wiki.java.net/bin/view/Mobileandembedded/HowTo05. Acesso em 27 out. 2008.