André Luiz Cardoso

UMA FERRAMENTA PARA OBTENÇÃO ON LINE DE CERTIFICADOS DIGITAIS PARA USO DE SMART CARDS EM APLICAÇÕES JAVA ME

Florianópolis

2007

André Luiz Cardoso

UMA FERRAMENTA PARA OBTENÇÃO ON LINE DE CERTIFICADOS DIGITAIS PARA USO DE SMART CARDS EM APLICAÇÕES JAVA ME

Trabalho de conclusão de curso apresentado à disciplina de Projetos II do curso de Ciência da Computação, Departamento de Informática e Estatística, Centro Tecnológico, Universidade Federal de Santa Catarina.

Florianópolis

2007

A presente Monografia foi aprovada como requisito para obtenção do grau de Bacharel em Ciências da Computação na Universidade Federal de Santa Catarina sob o título *"UMA FERRAMENTA PARA OBTENÇÃO ON LINE DE CERTIFICADOS DIGITAIS PARA USO DE SMART CARDS EM APLICAÇÕES JAVA ME"*, defendida por André Luiz Cardoso e aprovada em 2007, em Florianópolis, Estado de Santa Catarina, pela banca examinadora constituída por:

Ricardo Felipe Custódio
Orientador

Júlio da Silva Dias
Membro da banca

Marcelo Luiz Brocardo
Membro da banca

Sumário

Lista de Figuras

Resumo

A necessidade de tecnologias de criptografia no desenvolvimento de sistemas seguros para dispositivos móveis é um fato comprovado. Com o avanço dos recursos destes dispositivos, abre-se a possibilidade de se utilizar tecnologias como criptografia de chaves públicas, assinatura digital e demais aplicações de certificados digitais neste contexto, motivando o desenvolvimento de ferramentas e aplicações nesta área.

Já existem no mercado dispositivos móveis com suporte à certificados digitais. Estes dispositivos, normalmente já apresentam uma ferramenta para gerência de certificados digitais, ou seja, é possível instalar certificados de autoridades confiáveis, ou ainda importar certificados e chaves privadas pessoais, através do próprio sistema operacional do dispositivo. Porém, não se conhece uma ferramenta para Java ME que permita de forma simples a realização do processo de obtenção de certificados digitais de maneira on line.

A falta de tal ferramenta inviabiliza a obtenção de certificados para smart cards diretamente no dispositivo móvel, além de dificultar ainda mais o processo de obtenção de certificados, visto que existe a necessidade de transferir arquivos para o dispositivo.

Dentro deste contexto, este trabalho apresenta uma ferramenta para esta plataforma capaz de suprir essa necessidade, proporcionando a obtenção de certificados diretamente no dispositivo móvel, dispensando a importação de arquivos e permitindo o uso de smart cards.

Palavras-chave: ICP, Java ME, SATSA, Smart Cards, Dispositivos Móveis.

Abstract

The necessity of cryptography technologies on the development of secure systems for mobile devices is a comproved fact. With the evolution of the computional power of this devices, the possibility of using technologies like public key cryptography, digital signatures and other digital certificates applications in this context, motivates the development of tools and applications on this area.

Mobile devices with support on digital certificates already exists on the market. Normally, this devices already have a tool for digital certificates management, so it's possible to install certificates of trusted certificate authorities and to import personal digital certificates and private keys using the device operating system. However, a Java ME tool to on line execute the certificate enrollment process it is not known.

The lack of such tool makes impracticable to obtain a digital certificate for secure hardware elements such as smart cards directly on the mobile device. Beside that, without this tool, its necessary to transfer external files to the device, making the enrollment process more difficult.

In this context, this work presents a tool for the Java ME platform that is able to supply this necessity, making possible the on line certificate enrollment process execution directly on the mobile device, without the need of file import and allowing the use of smart cards.

Key-words: PKI, Mobile Devices, smart cards, Java ME, SATSA.

1 Introdução

O presente trabalho está inserido no contexto de módulos públicos para infra-estrutura de chaves públicas. O módulo público é a interface entre o cliente (usuário) e os componentes autoridade certificadora (AC) e autoridade de registro (AR). Neste sentido, o desenvolvimento de uma ferramenta de obtenção de certificados digitais para o uso de certificados digitais em dispositivos móveis, faz parte do esforço do LabSEC em estudar e desenvolver ferramentas de módulo público.

O uso da Internet em dispositivos móveis está em amplo crescimento. Com isso, somam-se todas as vantagens da rede mundial de computadores, tais como rapidez e facilidades para a troca de informações, às vantagens dos dispositivos móveis, como a portabilidade, facilidade de uso e alta popularidade.

Infelizmente, as desvantagens da Internet também são herdadas. Dentre as desvantagens da rede, os problemas de segurança são majoritários. A garantia de autenticidade, não repúdio, integridade e irretroatividade das informações trafegadas pela Internet são hoje, as questões principais relacionadas à segurança da informação. A criptografia proporciona diversas soluções para estes problemas, das quais se destacam a infra-estrutura de chaves públicas e os certificados digitais.

Os smart cards como dispositivos de armazenamento seguro são uma solução eficaz para proteger informações sensíveis como as chaves privadas, além disso, sua presença é comum nos dispositivos móveis (com destaque para os telefones celulares), porém o seu uso em aplicações ainda é restrito.

A plataforma Java Micro Edition é uma das mais utilizadas para o desenvolvimento de aplicações para dispositivos móveis. Porém, é notável a baixa popularidade de aplicações desta plataforma que utilizem estas soluções, especificamente no âmbito dos certificados digitais em conjunto com smart cards. Pode-se apontar como causa a dificuldade da obtenção on line de certificados digitais, devido à falta de ferramentas específicas.

1.1 Problema

Já existem no mercado dispositivos móveis com suporte à certificados digitais. Estes dispositivos, normalmente já apresentam uma ferramenta para gerência de certificados digitais, ou seja, é possível instalar certificados de autoridades confiáveis, ou ainda importar certificados e chaves privadas pessoais, através do próprio sistema operacional do dispositivo. Porém, não se conhece uma ferramenta para Java ME que permita de forma simples a realização do processo de obtenção de certificados digitais de maneira on line.

A falta de tal ferramenta inviabiliza a obtenção de certificados para smart cards diretamente no dispositivo móvel, além de dificultar ainda mais o processo de obtenção de certificados, visto que existe a necessidade de transferir arquivos para o dispositivo.

Dentro deste contexto, este trabalho propõe uma ferramenta para esta plataforma capaz de suprir essa necessidade, proporcionando a obtenção de certificados diretamente no dispositivo móvel, dispensando a importação de arquivos e permitindo o uso smart cards para aumentar o nível de segurança das aplicações Java ME.

1.2 Objetivos

1.2.1 Objetivo Geral

Baseando-se no problema apontado, este trabalho tem como objetivo geral a análise, projeto e desenvolvimento de uma ferramenta para a obtenção on line de certificados digitais para o uso de smart cards em dispositivos móveis que suportem a plataforma Java ME.

Com esta ferramenta, objetiva-se facilitar a adoção da tecnologia de certificação digital em conjunto com os dispositivos de armazenamento seguro em aplicações Java ME, agregando à estas as vantagens desta tecnologia que já é muito aplicada no contexto dos computadores pessoais.

Com o uso da ferramenta para obtenção de certificados digitais, usuários avançados (muitas vezes desenvolvedores), podem utilizá-los para a melhoria dos aspectos de segurança de suas aplicações Java ME, contando com o armazenamento seguro das chaves e certificados.

1.2.2 Objetivos Específicos

Para conclusão do objetivo geral é necessário atingir os seguintes objetivos:

- Descrever a plataforma Java ME e sua arquitetura;
- Fazer um levantamento das bibliotecas e frameworks para plataforma Java ME, para a realização de operações criptográficas e uso de dispositivos de armazenamento seguro;
- Levantar os processos de negócio envolvidos na obtenção de certificados digitais;
- Analisar, projetar e desenvolver a ferramenta cliente de uma ICP para solicitação e recebimento de certifiados digitais.

1.3 Justificativa

Os dispositivos móveis tais como telefones celulares e computadores de bolso vêm evoluindo e ganhando recursos computacionais que permitem a realização de funções antes restritas aos computadores pessoais tradicionais, tais como o acesso à web, correio eletrônico, comércio eletrônico e criptografia.

Segundo Andre N. Klingsheim, Veborn Moen e Kjell J. Hole (2007), o mercado deste tipo de dispositivo está crescendo rapidamente, demandando o desenvolvimento de software para estas aplicações.

A necessidade de melhorar os aspectos de segurança destas aplicações é a maior justificativa para aplicações como a proposta por este trabalho.

A melhora dos aspectos de segurança pode viabilizar muitas aplicações que têm como fator de sucesso requisitos de segurança. Alguns exemplos são aplicações ligadas ao governo, a justiça e bancárias, onde a autenticação do usuário é um fator crítico de sucesso.

1.4 Organização do Texto

Nesta seção será descrita a forma como o trabalho está organizado, indicando o conteúdo de cada um dos capítulos.

O primeiro capítulo trata da introdução, é definido o problema, os objetivos e a justificativa para o desenvolvimento do trabalho.

O segundo capítulo apresenta uma breve revisão dos conceitos relacionados encontrados na literatura, fundamentando o restante do trabalho.

No terceiro capítulo, é apresentada uma análise e descrição dos frameworks e bibliotecas

para operações criptográficas com o objetivo de escolher o que mais auxilia o desenvolvimento da ferramenta. Neste capítulo ainda se encontra uma conceituação da plataforma Java ME que serviu de base para o desenvolvimento da ferramenta proposta.

O quarto capítulo compreende a descrição do processo de desenvolvimento da ferramenta, detalhando as etapas percorridas e documentando os resultados obtidos. Cada caso de uso é detalhado, apresentando o comportamento da implementação de cada um deles.

Por fim, o quinto capítulo apresenta as conclusões, além de propor trabalhos futuros.

2 *Fundamentação Teórica*

Neste capítulo será realizada uma breve revisão da bibliografia com o objetivo de fundamentar e situar o trabalho com os conceitos relacionados.

Na primeira seção será apresentado o conceito de criptografia, tanto simétrica como assimétrica. Tais conceitos são importantes por serem ferramentas base para toda a área da segurança da informação.

Na segunda seção o conceito de infra-estrutura de chaves públicas será abordado. Este conceito é de fundamental importância para o entendimento da ferramenta desenvolvida, afinal é através de uma infra-estrutura de chaves públicas que a ferramenta facilita a obtenção de certificados digitais.

Na terceira seção o conceito de dispositivos móveis será contextualizado para este trabalho, apresentando o ambiente alvo da ferramenta.

A mesma contextualização feita em relação aos dispositivos móveis, é realizada para os smart cards na penúltima seção deste capítulo.

Finalizando o capítulo, uma conclusão sobre os conceitos abordados é apresentada, correlacionando-os com a ferramenta proposta.

2.1 Criptografia

Criptografia é a área da matemática e da computação que trata de problemas relacionados à segurança da informação. Diversas tecnologias atualmente aplicadas para garantir requisitos de autenticação, integridade, sigilo e não repúdio têm como base algoritmos criptográficos. Dentre estas, encontra-se a infra-estrutura de chaves públicas, tecnologia intimamente relacionada com a ferramenta de obtenção on line de certificados digitais que é objeto deste trabalho.

William Stallings (1998) define a criptografia da seguinte maneira:

> "consiste na arte de escrever em cifras e códigos, fazendo uso de um conjunto de técnicas que tornam uma mensagem incompreensível (texto cifrado). Este processo é conhecido como cifragem. A decifragem, que é o processo inverso, consiste em transformar o texto cifrado em texto compreensível (texto plano), de tal forma que somente o destinatário consegue a informação."

A palavra criptografia tem origem grega e pode ser traduzida como escrita escondida. Basicamente, a criptografia trata de algoritmos para embaralhar e desembaralhar mensagens garantindo o sigilo da mensagem transmitida entre duas ou mais entidades comunicantes.

Além do sigilo, outros aspectos de segurança são tratados pela criptografia:

- Sigilo - diz respeito à confiança de que somente pessoas autorizadas terão acesso à informação;
- Autenticidade - diz respeito à garantia de que quem enviou a informação é realmente quem deveria ter enviado;
- Integridade - garantia de que a informação não foi alterada;
- Não-repúdio - garantia de que quem enviou a informação não pode negar o seu envio;
- Tempestividade - garante que a informação existe e está íntegra a partir de uma determinada data.

Estes aspectos são garantidos por diversas técnicas e algoritmos da criptografia. Diante destes fatos, é de fundamental importância uma breve revisão de conceitos fundamentais da criptografia como será apresentado nos próximos tópicos.

2.1.1 Criptografia simétrica

A criptografia simétrica também é conhecida como criptografia tradicional. A principal característca dessa técnica é o fato de se utilizar uma única chave secreta compartilhada entre os comunicantes.

O remetente e o destinatário utilizam o mesmo valor para a chave. O remetente utiliza um algoritmo de cifragem que com o valor da chave, embaralha o texto produzindo o texto cifrado. O destinatário por sua vez, utiliza o mesmo algoritmo e a mesma chave para decifrar o texto cifrado, obtendo assim o texto plano (original) (Russ Housley; Tim Polk, 2001, p. 6).

Portanto, a criptografia simétrica exige que tanto o remetente quanto o destinatário compartilhem a chave e o algoritmo utilizados. Em função disto, o compartilhamento da chave se torna um problema em comunicações através da internet, pois a informação da chave utilizada na comunicação trafega em aberto pela rede, possibilitando um intruso obter a chave.

2.1.2 Criptografia assimétrica

Esta técnica também é conhecida como criptografia de chave pública, nela utiliza-se um par de chaves ao invés de uma única como na criptografia simétrica. Uma das chaves, denominada chave privada é mantida sob sigilo. A outra, chamada de chave pública deve ser de alguma forma divulgada.

Qualquer das chaves pode ser usada por algoritmos criptográficos, sendo que, se a chave privada for usada pra cifrar o documento, somente a chave pública poderá decifrá-lo, e vice-versa (William Stallings, 1998).

A criptografia de chave pública foi proposta por Whitfield Diffie e Martin Hellman (1976) em 1976, e um ano depois, o primeiro algoritmo a utilizar esta técnica, o RSA, foi apresentado. Proposto por R. L. Rivest, A. Shamir e L. M. Adelman (1977), o RSA é ainda hoje base de grande parte das aplicações que utilizam criptografia assimétrica.

Através do par de chaves, o problema do compartilhamento e distribuição da chave secreta da criptografia simétrica é eliminado. Porém, o custo computacional para a realização das operações criptográficas dos algoritmos de criptografia assimétrica é maior que os de criptografia simétrica. Na prática, é comum utilizar-se das duas técnicas aliando segurança e desempenho.

Apesar de resolver o problema do compartilhamento da chave secreta da criptografia simétrica, a técnica da criptografia assimétrica lança um problema em relação ao modo de distribuição das chaves públicas.

As chaves públicas devem ser disponibilizadas a todos, porém de forma segura e confiável. Além disso, deve existir um meio de se associar uma chave pública, que nada mais é do que uma sequência de bits, a uma entidade real. Indo mais além, deve ser possível identificar a validade e aplicações possíveis dessa chave.

Como solução para estes problemas, encontra-se na literatura duas diferentes abordagens:

- Rede de confiança;
- Infra-estrutura de chaves públicas;

Devido a ampla utilização, inclusive a adoção por entidades governamentais e grandes companhias, a Infra-estrutura de chaves públicas tem uma maior aceitação, portanto, foi escolhida como a abordagem utilizada neste trabalho e será apresentada com mais detalhes na próxima seção.

2.2 Infra-estrutura de Chaves Públicas

Na abordagem da Infra-estrutura de chaves públicas existe o conceito de certificado digital, que de uma maneira bem simplificada é uma estrutura de dados que amarra dados do usuário tais como nome, país, organização à sua chave pública.

O sistema da infra-estrutura de chaves públicas se encarrega de emitir certificados digitais que de forma confiável, associa entidades à suas chaves públicas. Para isso, é criada uma hierarquia de autoridades, que em suma são entidades das quais qualquer pessoa pode confiar, estas entidades são análogas à instituições como cartórios, órgãos governamentais, etc.

Dentre estas entidades, existem as autoridades certificadoras e autoridades de registro. As autoridades certificadoras são entidades que possuem seus próprios certificados digitais e têm a responsabilidade de emitir certificados digitais para os usuários finais. Atuam como terceira parte confiável da comunicação.

Para identificar se um certificado digital e sua chave pública é realmente o certificado da entidade que se quer comunicar, verifica-se se o certificado que emitiu (assinou) o seu certificado é de uma autoridade certificadora confiável. Desta forma, a autoridade certificadora atua como terceira parte confiável, na qual ambas as partes que se comunicam confiam, resolvendo o problema da autenticação das chaves públicas.

A outra entidade das infra-estruturas de chaves públicas é a autoridade de registro. Esta autoridade é responsável por registrar os usuários, ou seja, ela quem realmente desempenha a verificação dos dados do usuário que solicita um certificado digital. Para isto, cada autoridade de registro deve seguir um conjunto de regras e procedimentos descritos no documento entitulado "Declaração de Práticas de Certificação" o qual deve ser disponibilizado pela infra-estrutura de chaves públicas.

A autoridade de registro é uma entidade especializada na realização do processo de registro e é utilizada quando há muita demanda por registros para aumentar a escalabilidade da infra-estrutura de chaves públicas e reduzir custos (Carlisle Adams; Steve Lloyd, 2002). Não é imprescindível para o funcionamento da infra-estrutura de chaves públicas como é a autoridade

certificadora. Caso o volume de certificados a serem emitidos não seja muito grande, a própria autoridade certificadora pode assumir o papel da autoridade de registro.

As próximas seções tratam dos dispositivos móveis e smart cards. A equalização dos conceitos destes elementos no âmbito deste trabalho é de fundamental importância para o entendimento da ferramenta proposta neste trabalho.

2.3 Dispositivos Móveis

Quando se fala de dispositivos móveis, normalmente logo se pensa nas restrições de conectividade, processamento e memória que são comuns a estes dispositivos. Porém, estas restrições vêm diminuindo com a melhoria dos recursos computacionais dos dispositivos, o que transforma os dispositivos móveis em aparelhos multifuncionais, capazes de realizar funcionalidades de entretenimento, acesso à internet, acesso à serviços financeiros, serviços de navegação e informação, entre outros (BOK, 2006).

Diante deste contexto de evolução, entede-se por dispositivos móveis, no âmbito deste trabalho, aparelhos portáteis capazes de acessar a internet, mais especificamente, telefones celulares e PDA's que apresentam esta capacidade. Além de serem capazes de acessar a internet, os dispositivos alvos deste trabalho devem permitir a utilização de certificados digitais e operações criptográficas com o uso de smart cards.

Estes dispositivos têm muita popularidade e portanto têm uma necessidade maior de aplicações seguras. O fato de terem uma grande proximidade com smart cards também contribui para serem alvo da ferramenta proposta neste trabalho.

Para facilitar o desenvolvimento de aplicações para dispositivos móveis, existem algumas plataformas capazes de abstrair detalhes e peculiaridades de cada modelo destes aparelhos. Dentre estas plataformas, destacam-se BREW e Java ME.

O grupo de pesquisa Zelos Group comparou as duas plataformas, indicando que a plataforma Java ME tem as seguintes vantagens (MCATEER, 2004):

- Maior número de dispositivos que a suportam. Cerca de 63 milhões em 2009 contra 38 milhões suportando BREW;
- Envolvimento de vários fabricantes na definição e extensão da plataforma;
- Crescimento melhor planejado em relação a BREW;

Dentro do objetivo de prover segurança utilizando certificados digitais, na produção de documentos eletrônicos confiáveis, a plataforma Java ME também se torna mais vantajosa devido às diversas bibliotecas, especificações e frameworks existentes, inclusive para operações de segurança. Por estes motivos, a plataforma Java ME foi escolhida para o desenvolvimento da ferramenta de obtenção on line de certificados digitais.

Assim como nesta seção o conceito de dispositivos móveis foi trazido para o contexto deste trabalho, na próxima seção será contextualizado o conceito de smart card.

2.4 Smart cards

Smart cards são dispositivos muito similares aos cartões de crédito, porém, contém capacidade computacional, pois possuem uma CPU, memória e sistemas operacionais embarcados (RSA, 2001).

A maior característica dos smart cards criptográficos é a segurança no armazenamento dos dados, devido aos fatores físicos e mecânicos, além do fato de fisicamente ser portável e pessoal, facilitando a manutenção e proteção dos dados que carrega.

Além de serem projetados para armazenar chaves privadas e certificados, os smart cards executam operações criptográficas com as chaves que armazena. Dessa maneira, as chaves nunca saem do dispositivo, o que traz uma maior segurança.

Outro fator importante a ser levado em consideração é o fato do mercado de smart cards estar intimamente ligado ao mercado dos dispositivos móveis utilizados no setor de telecominicações. Segundo Ward (2001), este mercado é o maior comprador de smart cards.

Ainda sobre a relação dos dispositivos móveis conceituados na seção anterior com os smart cards, destaca-se a informação de que mais de 50% dos Java cards (smart cards que rodam Java) fabricados, estão no setor de comunicação móvel (JAVA CARD FORUM, 2007).

2.5 Conclusão

Após a revisão dos conceitos relacionados a este trabalho, pode-se concluir que os conceitos da criptografia, tanto simétrica quando assimétrica, são base para as aplicações de segurança da informação.

A infra-estrutura de chaves públicas é uma solução muito bem aceita pelo mercado, por governos e instituições, motivando o desenvolvimento de ferramentas correlacionadas.

Os dispositivos móveis vêm ganhando recursos computacionais que possibilitam o aparecimento de aplicações mais avançadas, principalmente com o uso da plataforma Java ME. Os smart cards fornecem armazenamento seguro e operações criptográficas, além de estarem muito bem distribuídos no mercado dos dispositivos móveis.

A elevação do nível de segurança das aplicações Java ME é um objetivo possível de ser alcançado com as tecnologias existentes e o desenvolvimento de ferramentas como a proposta por este trabalho contribui para alcançá-lo.

3 *Levantamento de frameworks e bibliotecas criptográficas*

No capítulo anterior (Capítulo 2), concluiu-se que o nível de segurança das aplicações Java ME é um objetivo possível de ser alcançado com as tecnologias existentes, justificando o desenvolvimento da ferramenta proposta. Sendo assim, neste capítulo serão descritos os frameworks e bibliotecas identificados e analisados com o intuito de suportar e facilitar o desenvolvimento da ferramenta.

Os critérios de análise utilizados foram: o atendimento do requisito da manipulação de chaves e certificados digitais, o acesso à smart cards, a facilidade de uso, além da gratuidade da biblioteca (licença e preço).

Na primeira seção, a plataforma Java ME será descrita, facilitando o entendimento dos frameworks e bibliotecas específicas descritos nas seções seguintes.

A segunda seção apresenta o framework SATSA descrevendo as funcionalidades fornecidas, a arquitetura típica e os pacotes disponibilizados.

A terceira seção descreve sucintamente a biblioteca Bouncy Castle Lightweight, apresentando suas vantagems e desvantagens em relação aos critérios apresentados.

A penúltima seção apresenta a biblioteca IAIK, também apresentando as vantagems e desvantagens do uso desta biblioteca para o desenvolvimento da ferramenta de obtenção on line de certificados digitais.

Por fim, a última seção apresenta uma conclusão sobre as bibliotecas analisadas, indicando a que mais atende aos critérios selecionados para a escolha da biblioteca a ser utilizada.

3.1 Plataforma Java ME

A tecnologia Java compreende três elementos: a linguagem de programação utilizada para escrever as aplicações; a máquina virtual, utilizada para executar as aplicações; e um vasto conjunto de bibliotecas, frameworks e APIs que apoiam o desenvolvimento de aplicações, promovendo o reuso de funcionalidades já implementadas (SM, 2007).

Buscando atingir o máximo de portabilidade da tecnologia Java, a Sun Microsystems a particionou em três edições, listadas por Yuan (2003) , da seguinte maneira:

- Java Standard Edition (Java SE): É a base da tecnologia Java. Define a máquina virtual e as bibliotecas que rodam em computadores pessoais e estações de trabalho;
- Java Enterprise Edition (Java EE): É formada pela Java Standard Edition com várias bibliotecas, APIs, containers e ferramentas específicas para rodar aplicações em servidores;
- Java Micro Edition (Java ME): É projetada para pequenos dispositivos, contém máquinas virtuais leves e especialmente projetadas para estes dispositivos. É formada por um subconjunto das bibliotecas da Java SE e por bibliotecas específicas.

A relação entre as edições está representada na Figura 3.1:

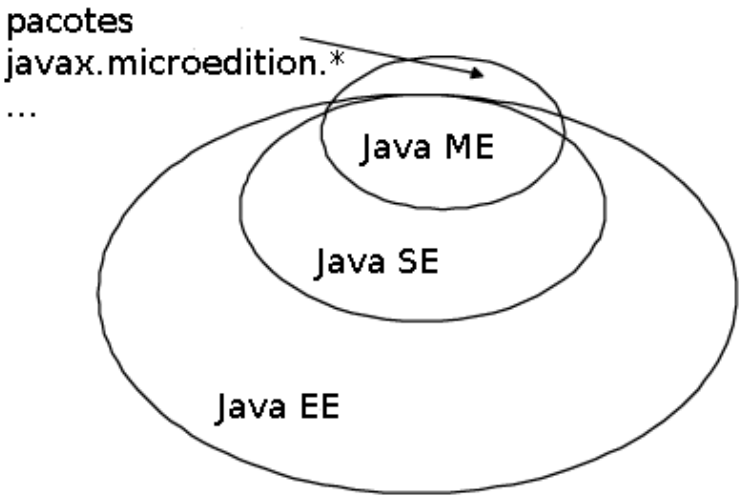

Figura 3.1: Edições da plataforma Java. Adaptada de (YUAN, 2003)

Pode-se notar que a Java ME contém um subconjunto da Java SE. Porém, somente restringir as APIs e bibliotecas e associá-las em uma nova edição não garante a portabilidade no universo de pequenos dispositivos.

Como Ortiz (2004) define:

> "J2ME não define uma nova linguagem de programação; ela adapta a tecnologia Java existente para handhelds e dispositivos embarcados. J2ME mantém a compatibilidade com J2SE aonde for possível. Para resolver as limitações dos pequenos dispositivos, J2ME às vezes substitui APIs da J2SE e adiciona novas interfaces..."

Existe uma grande variedade de pequenos dispositivos, desde pagers, celulares e PDAs até dispositivos que se assemelham aos computadores pessoais como dispositivos de TV digital (KNUDSEN, 2003). Dentro de cada uma dessas categorias, as possibilidades e recursos variam de dispositivo para dispositivo.

Para manter a portabilidade e não perder a possibilidade de utilização de recursos que não sejam comuns a todos os dispositivos, a plataforma Java ME foi organizada em configurações, perfis e pacotes opcionais. A Figura 3.2 ilustra a relação destes componentes.

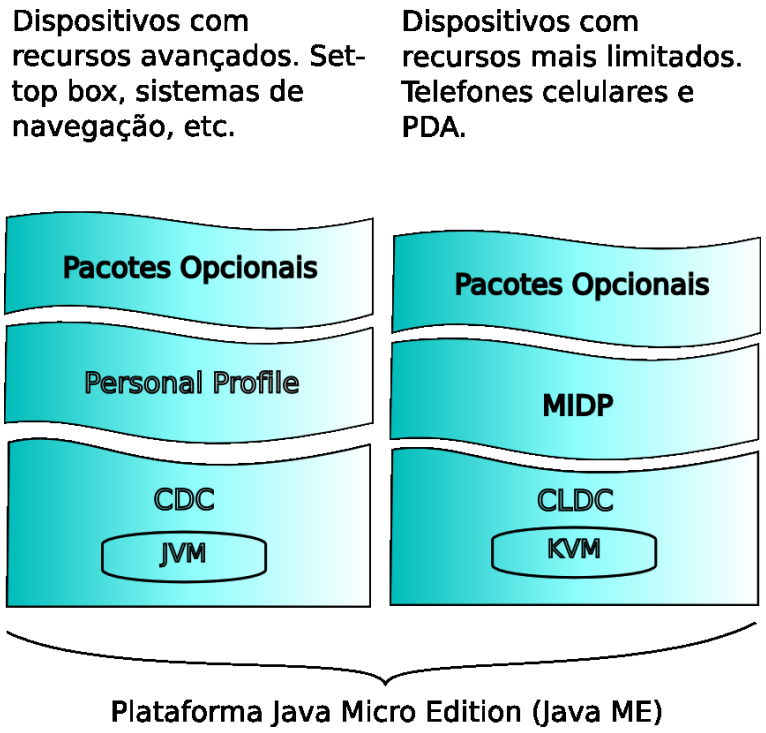

Figura 3.2: Componentes da tecnologia Java ME. Adaptada de (SUN MICROSYSTEMS, 2007)

Nas próximas subseções, cada componente presente na Figura 3.2 será apresentado.

3.1.1 Configurações

Para manter a portabilidade dentro de todos os tipos de dispositivos móveis, foram definidas configurações dentro do Java ME. Dessa maneira as configurações são definidas de acordo com as restrições de memória e processamento de cada família de dispositivos (KNUDSEN, 2003).

Segundo Ortiz (2004):

> "Uma configuração define um ambiente de execução J2ME básico, menor-denominador-comum. Esse ambiente inclui uma máquina virtual e um conjunto do núcleo de classes primariamente derivados da J2SE."

Hoje existem definidas duas configurações: CDC (Connected Device Configuration) e CLDC (Connected Limited Device Configuration).

Configuração CDC

A configuração CDC foi desenvolvida para suportar pequenos dispositivos com uma capacidade computacional suficientemente grande para rodar uma máquina virtual java completa como a da Java SE. Os exemplos mais comuns desse tipo de dispositivo são os set-top boxes de televisão, sistemas de navegação para carros e PDAs com alto processamento (KNUDSEN, 2003).

Apesar de definir uma máquina virtual completa como a da Java SE, a configuração CDC não especifica as bibliotecas existentes no Java SE da mesma maneira. Estas sofrem modificações para atender as necessidades das restrições de processamento e memórias que os pequenos dispositivos apresentam. Assim, algumas bibliotecas têm suas interfaces modificadas e outras foram totalmente removidas. O resultado disso é um ambiente de execução Java que se encaixa em dispositivos que tenham no mínimo 2 MB de memória RAM e 2 MB de memória ROM (SUN MICROSYSTEMS, 2005).

Para os dispositivos que não apresentam estes recursos, foi definida a configuração CLDC, apresentada na subseção seguinte.

Configuração CLDC

O objetivo principal da CLDC é atender os requisitos de tamanho de memória e recursos computacionais de telefones celulares e outros pequenos dispositivos(SUN MICROSYSTEMS, 2005). Como visto no capítulo anterior, estes são os dispositivos alvo deste trabalho, pelo fato de terem uma maior popularidade e por esta razão terem uma maior necessidade de aplicações seguras. Além disso, os telefones celulares têm um contato muito grande com smart cards, o que possibilita a utilização destes para elevar o nível de segurança das aplicações Java ME.

Os dispositivos suportados pela CLDC são equipados com processadores de 16 bits ou 32 bits com pelo menos 160 kilobytes de memória não volátil e 32 kilobytes de memória volátil,

totalizando 192 kilobytes. Além disso, o consumo de energia destes dispositivos é baixo, pois são tipicamente alimentados por baterias e é comum que a conexão com a rede seja intermitente e de baixa velocidade (ORTIZ, 2004).

Diferentemente da CDC, a CLDC não especifica uma máquina virtual completa, ao invés disso, é especificada uma máquina virtual reduzida, chamada de KVM. Este nome se deve ao fato do tamanho de seu tamanho ser medido em kilobytes e não em megabytes como nas outras máquinas virtuais (KNUDSEN, 2003).

3.1.2 Perfis

Um perfil é uma forma de se organizar APIs necessárias para se desenvolver aplicações específicas para uma família de dispositivos. O perfil situa-se uma camada acima da configuração, ou seja, é especificado para rodar sobre uma configuração.

A organização das APIs da plataforma em perfis, provê flexibilidade aos desenvolvedores ao suportar diferentes tipos de dispositivos sobre um mesmo ambiente de execução (definido pela configuração). Os perfis fazem com que a plataforma Java ME apresente opções de APIs para cada categoria de dispositivos, ao invés de definir uma única API que contemple a maioria das situações, como é feito na plataforma Java SE.

Cada perfil é preparado para ser executado sobre uma configuração. Atualmente, o perfil mais utilizado para a configuração CDC é o Personal Profile que define um subconjunto das APIs presentes no Java SE.

Para a configuração CLDC, o perfil Mobile Information Device Profile (MIDP) é o mais utilizado. Este perfil facilidades para o desenvolvimento de aplicações para a configuração CLDC, tais como as definidas em (SUN MICROSYSTEMS, 2006):

- ciclo de vida de aplicações, definindo a semântica das aplicações MIDP e como elas são controladas;
- funcionalidades de rede;
- persistência de dados;
- segurança fim-a-fim através do protocolo HTTPS;
- interface gráfica, incluindo facilidades para a entrada e saída de dados.

3.1.3 Pacotes Opcionais

As configurações e os perfis definem a base para o desenvolvimento de aplicações móveis. Seu objetivo é agregar funcionalidades comums à famílias de dispositivos. Os pacotes opcionais, por sua vez, definem as operações específicas necessárias em aplicações específicas.

Giguere (2002) relaciona os pacotes opcionais com os perfis da seguinte maneira:

> "Um pacote opcional é também um conjuto de APIs, porém direfentemente de um perfil, não define um ambiente de aplicação completo. Um pacote opcional sempre é usado em conjunto com uma configuração ou um perfil. Ele estende o ambiente de execução para suportar as capacidades dos dispositivos que não são universais o suficiente para serem definidas como parte do perfil ou que precisem ser compartilhadas por perfis diferentes. (GIGUERE, 2002)"

Assim como as configurações e os perfis, os pacotes opcionais são padronizados pela comunidade Java, com o apoio da Sun Microsystems e de outros fabricantes. Por conta disso, são suportados diretamente pelos dispositivos que os disponibiliza aos desenvolvedores. Isto trás uma grande vantagem para os desenvolvedores pois faz com que as aplicações Java ME se mantenham com um tamanho reduzido quando utilizam apenas componentes padronizados.

Nas seções seguintes serão apresentados os pacotes opcionais definidos pela Security and Trust Services API (SATSA). Além destes pacotes opcionais, as bibliotecas externas Bouncy Castle e IAIK também serão abordadas com o objetivo de identificar a biblioteca ou framework que melhor suporta o desenvolvimento da ferramenta de obtenção on line de certificados digitais.

3.2 SATSA

Dentre os pacotes opcionais existentes, os que tem maior relevência para este trabalho são os definidos pela Security and Trust Services API (SATSA) especificada no documento JSR 177. Estes pacotes opcionais definem APIs que provêm as seguintes funcionalidades:

- assinatura digital no nível da aplicação;
- gerenciamento básico de credenciais de usuários;
- operações criptográficas; e

- comunicação com smart cards (SUN MICROSYSTEMS,).

O objetivo destas APIs é prover serviços de segurança e confiança, baseando-se no conceito de elemento seguro (security element - SE). Um selemento seguro é definido como um componente em um dispositivo J2ME que fornece:

- armazenamento seguro para proteger dados como chaves privadas, certificados raizes, informações pessoais, entre outros;
- operações criptográficas para garantia da integridade e confidencialidade de dados;
- um ambiente de execução seguro para funcionalidades de segurança (SUN MICROSYSTEMS, 2004b).

A especificação do SATSA não limita a implementação do elemento seguro em smart cards, apesar de alguns pacotes serem otimizados para este tipo de dispositivo. Além dos smart cards, um elemento seguro pode ser implementado pelo próprio dispositivo ou ainda totalmente em software (SUN MICROSYSTEMS, 2004b). Porém, dentro do contexto deste trabalho, assume-se que o elemento seguro utilizado seja um smart card.

Como a especificação não se baseia em nenhuma implementação específica, um dispositivo pode ter vários elementos seguros disponíveis. O tipo do elemento seguro utilizado fica transparente ao desenvolver, pois a implementação do SATSA que trata a interação com o elemento seguro (Andre N. Klingsheim; Veborn Moen; Kjell J. Hole, 2007).

A SATSA define 4 pacotes opcionais:

- SATSA-APDU
- SATSA-JCRMI
- SATSA-PKI
- SATSA-CRYPTO

A Figura 3.3 demonstra a disposição destes pacotes em uma arquitetura de uma aplicação MIDP típica que utilize o SATSA.

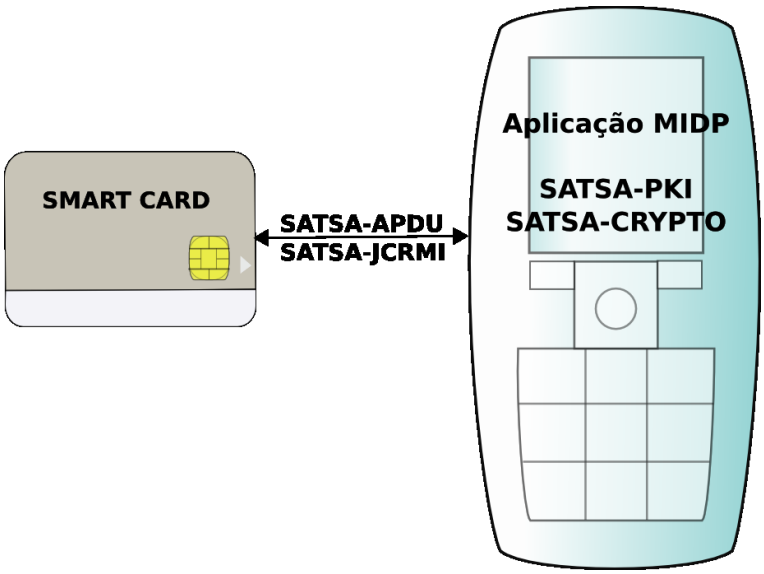

Figura 3.3: Arquitetura típica de uma aplicação MIDP que utiliza SATSA

O SATSA-APDU e o SATSA-JCRMI tratam da comunicação entre a aplicação J2ME e o smart card. Esta comunicação se baseia em um modelo cliente-servidor de requisições e respostas síncronas onde a aplicação MIDP é o cliente e a aplicação que roda no smart card é o servidor (ORTIZ, 2005).

Na SATSA-APDU, o protocolo utilizado é o definido pela ISO 7816-4 APDU. Este protocolo é considerado de baixo nível em relação ao utilizado no SATSA-JCRMI. Este, por sua vez, adota uma abordagem orientada a objetos utilizando a invocação remota de métodos (RMI) para enviar os comandos e receber as respostas. A utilização do SATSA-JCRMI é encorajada por sua facilidade de uso em contrapartida ao SATSA-APDU. Este deve ser utilizado quando o smart card não suportar o SATSA-JCRMI, ou seja, quando não implementar a especificação Java Card (SUN MICROSYSTEMS, 2004a).

O pacote SATSA-PKI provê serviços de assinatura digital e gerenciamento de credenciais de usuários. O serviço de assinatura digital permite que uma aplicação Java ME gere assinaturas digitais em conformidade com o padrão Cryptographic Message Syntax (CMS) definido em (HOUSLEY, 2004) (RFC 3852).

O serviço de gerenciamento de credenciais provê à aplicação Java ME um mecanismo para criação de requisições de certificados digitais (CSR) em conformidade com o padrão PKCS#10 definido em (M. Nystrom; B. Kaliski, 2000) (RFC 2986). Além disso, possibilita a instalação e remoção de certificados digitais do repositório de certificados implementado no smart card (SUN MICROSYSTEMS, 2004b).

As funcionalidades fornecidas pelo SATSA-PKI baseiam-se nas operações criptográficas providas pelo smart card, assim, os pacotes de comunicação SATSA-APDU ou SATSA-JCRMI são utilizados de maneira transparente. O contato com o smart card em algumas operações exige interação com o usuário, como na solicitação de senha ou PIN (Personal Identification Number), portanto, o SATSA pode exibir telas para este tipo de interação, livrando o desenvolvedor dessa tarefa.

O pacote SATSA-CRYPTO provê funcionalidades para a realização de operações criptográficas de propósito geral. Este pacote disponibiliza um subconjunto da API de criptografica do Java SE versão 1.4.2. Este subconjunto possibilita as seguintes operações: cálculo de resumos criptográficos (hash), verificação de assinaturas, crifragem e decifragem (SUN MICROSYSTEMS, 2004b).

Portanto, para o utilizador do SATSA não é necessário se preocupar em como acessar e armazenar dados no smart card, pois o framework SATSA já assume essa responsabilidade e a implementa de acordo com o dispositivo alvo. Portanto, este framework se torna muito interessante para aplicações que tenham como requisito a utilização de smart cards, como é o caso da ferramenta desenvolvida neste trabalho.

3.3 Bouncy Castle Lightweight API

The Legion of Bouncy Castle iniciou como uma iniciativa para a implementação de um provedor de criptografia para o Java Cryptographic Extension (JCE) totalmente gratuito e de código aberto (YUAN, 2003). A implementação foi organizada de tal forma que pode ser empacotada tanto para Java SE como para Java ME.

Por ser um projeto de código aberto, esta biblioteca apresenta uma série de vantagens, enumeradas por Yuan (2003) na seguinte lista:

- Bugs ou falhas de segurança são corrigidos rapidamente;
- As APIs foram projetadas de forma flexível, e juntamente com o modelo de desenvolvimento comunitário, permitem que qualquer desenvolvedor possa contribuir com novas implementações de algoritmos. Conseqüentemente, a biblioteca implementa um grande número de algoritmos conhecidos;
- A comunidade de desenvolvedores está sempre otimizando as implementações existentes, garantindo a melhoria do desempenho das operações criptográficas;

- É de graça.

Esta biblioteca suporta uma diversidade muito grande de algoritmos, sendo bem completa e funcional, apresentando muito mais opções de algoritmos e operações criptográficas (hash, por exemplo) quando comparada com os pacotes opcionais da SATSA. Porém, ao contrário da SATSA, não é padronizada como pacotes opcionais, não sendo portanto, suportada nativamente pelos dispositivos, o que ocasiona um aumento significativo do tamanho da aplicação Java ME, pois esta tem que incluir em seu pacote as classes da Bouncy Castle utilizadas.

Outra desvantagem é o fato de não dar suporte ao uso de smart cards, além de carecer de uma documentação eficaz, principalmente no âmbito do pacote para Java ME (lightweight).

3.4 IAIK

The Institute for Applied Information Processing and Communications (IAIK) é um provedor austríaco de soluções comerciais de criptografia e segurança baseadas em Java.

Dentro destas soluções, como uma biblioteca possível de ser utilizada neste trabalho, destaca-se a IAIK JCE Micro Edition, uma bibliteca criptográfica leve específica para a plataforma Java ME.

Assim como a Bouncy Castle, esta biblioteca oferece uma variedade de funcionalidades criptográficas, incluindo funções de hash, algoritmos de criptografia simétrica e assimétrica, cifragem de blocos e fluxo e gerenciamento de chaves e certificados (IAIK, 2007).

É uma biblioteca que apresenta um bom desempenho e tem um tamanho muito reduzido. Segundo Yuan (2003),

> "o arquivo jar tem somente 100 Kb de tamanho, que pode ser ainda mais reduzido no momento de empacotamento, se a aplicação não utilizar todas as funcionalidades."

Suas desvantagens se assemelham as da Bouncy Castle, pois também não dá suporte ao uso de smart cards e não ser padronizada como pacote opcional. Além disso, o fato de ser comercial também pode ser apontado como uma desvantagem.

3.5 Conclusão

A plataforma Java ME é uma solução muito inteligente e bem sucedida para o problema da portabilidade entre os mais variados aparelhos existentes no mercado de dispositivos móveis. Dentre os componentes existentes, os que mais se aproximam dos dispositivos conceituados no Capítulo 2 são: a configuração CLDC e o perfil MIDP.

Dentre as bibliotecas e frameworks existentes para a realização de operações criptográficas, a única que suporta o acesso a smart cards é a SATSA. Além disso, este pacote opcional é gratuito e permite a manipulação de chaves e certificados digitais. Portanto, os pacotes opcionais definidos pela SATSA são os que melhor atendem os requisitos da ferramenta, e por conta disso foram utilizados no desenvolvimento desta, que será apresentado no Capítulo 4.

4 *Desenvolvimento*

No capítulo anterior (Capítulo 3), identificou-se os componentes Java ME e o framework de operações criptográficas que melhor atendem o desenvolvimento da ferramenta. Neste capítulo será apresentado como tais componentes foram utilizados no desenvolvimento da ferramenta de obtenção on line de certificados digitais para uso de smart cards em aplicações Java ME.

A primeira seção descreve os processos de negócio envolvidos na obtenção de certificados digitais, identificando-se os atores envolvidos e as atividades desempenhadas.

Com base na análise dos processos de negócio, os requisitos são identificados e apresentados na seção seguinte.

A terceira seção apresenta as ferramentas utilizadas no desenvolvimento do trabalho.

A quarta seção descreve a ferramenta desenvolvida, apresentando o seu funcionamento e detalhando o seu projeto. A execução de cada caso de uso é descrita, apresentando o comportamento da ferramenta.

A quinta seção discute os resultados obtidos com o desenvolvimento da ferramenta.

Por último, é apresentada uma conclusão a respeito do desenvolvimento.

4.1 Análise de processos de negócio

Para a identificação dos requisitos da aplicação, é necessário um entendimento do processo de negócio a ser implementado por tais requisitos. Para tal fim, nesta seção serão documentados os processos envolvidos na obtenção de certificados digitais.

Nas infra-estruturas de chaves públicas mais conhecidas, é comum que as funções de registro e certificação serem desempenhadas em um único processo. Isto é, o usuário se relaciona apenas uma vez com as autoridades da infra-estrutura para obter o seu certificado.

No contexto dos dispositivos móveis, é interessante que a obtenção de certificados digitais

ocorra em dois processos separados. Uma das razões para isto é a dificuldade de preenchimento de formulários nas telas destes dispositivos, o que inviabiliza a execução direta dos processos de registro e certificação de maneira on line.

Nestes processos, existem sempre duas entidades envolvidas, o usuário e a autoridade certificadora. Em algumas atividades, dependendo da política utilizada pela infra-estrutura de chaves públicas, a autoridade certificadora pode ser substituída pela autoridade de registro.

Como visto no capítulo da fundamentação teórica (2), a autoridade de registro é uma entidade especializada na realização do processo de registro e é utilizada quando há muita demanda por registros para aumentar a escalabilidade da infra-estrutura de chaves públicas e reduzir custos (Carlisle Adams; Steve Lloyd, 2002).

Nas subseções a seguir serão apresentados os processos de registro e de certificação, ilustrados na forma de diagrama de atividades.

4.1.1 Processo de registro

O objetivo deste processo é o de atestar a identidade do usuário, julgando se a autoridade certificadora deve ou não emitir o certificado. O nível de detalhes dos critérios utilizados na verificação da identidade do usuário é depentende das políticas e práticas de certificação da infra-estrutura de chaves públicas em questão. Portanto, algumas etapas podem não ocorrer de maneira on line (Carlisle Adams; Steve Lloyd, 2002).

A Figura 4.1 ilustra um processo de registro típico.

Primeiramente, o usuário identifica-se perante a autoridade certificadora ou autoridade de registro, com o objetivo de comprovar a sua identidade. Nesse processo é comum, dependendo da política e prática de certificação, a identificação presencial com apresentação de documentos. Em um processo totalmente online, a identificação do usuário poderia ocorrer através o preenchimento de formulários em um browser.

Após receber a identificação, a autoridade certificadora ou autoridade de registro avalia o pedido de registro, verifica a identidade do usuário e efetua o registro caso a identidade tenha sido confirmada. Em um processo on line, esta atividade pode ser efetuada com o cadastro do usuário em um sistema de apoio ao processo de registro. Finalizando o proceso, a autoridade envia a confirmação do registro à entidade, possibilitando a execução do processo de certificação.

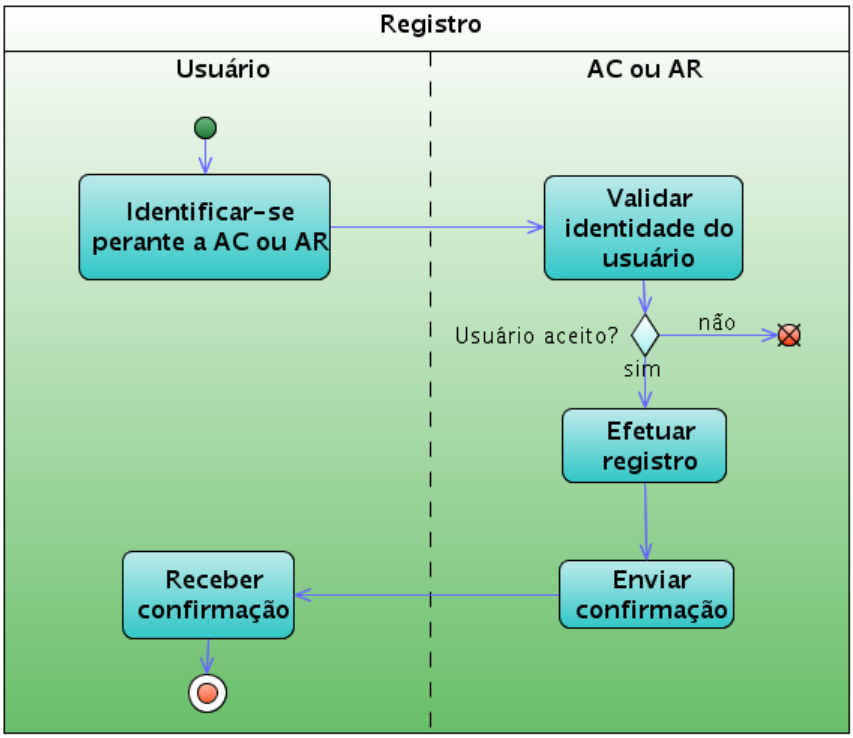

Figura 4.1: Processo de registro

4.1.2 Processo de certificação

Segundo Carlisle Adams e Steve Lloyd (2002), o processo de ligar a identidade de uma entidade final a uma chave pública através da emissão de um certificado digital realizado por uma autoridade certificadora é chamado de certificação.

A Figura 4.2 ilustra um processo típico de certificação. A ferramenta de obtenção de certificados digitais desenvolvida neste trabalho, implementa as atividades do usuário descritas nesta seção.

Este processo inicia-se com usuário gerando o seu par de chaves. Normalmente, a chave privada fica armazenada em um repositório seguro, implementado em software ou em hardware como por exemplo nos smart cards.

Após a geração do par de chaves, o usuário gera uma requisição de certificado. Esta requisição normalmente contém a chave pública, dados de identificação do usuário (nome, e-mail, país, estado, etc), bem como uma prova de que o usuário possui a chave privada relacionada à chave pública. Uma maneira simples de se provar isso é através da assinatura digital da requisição.

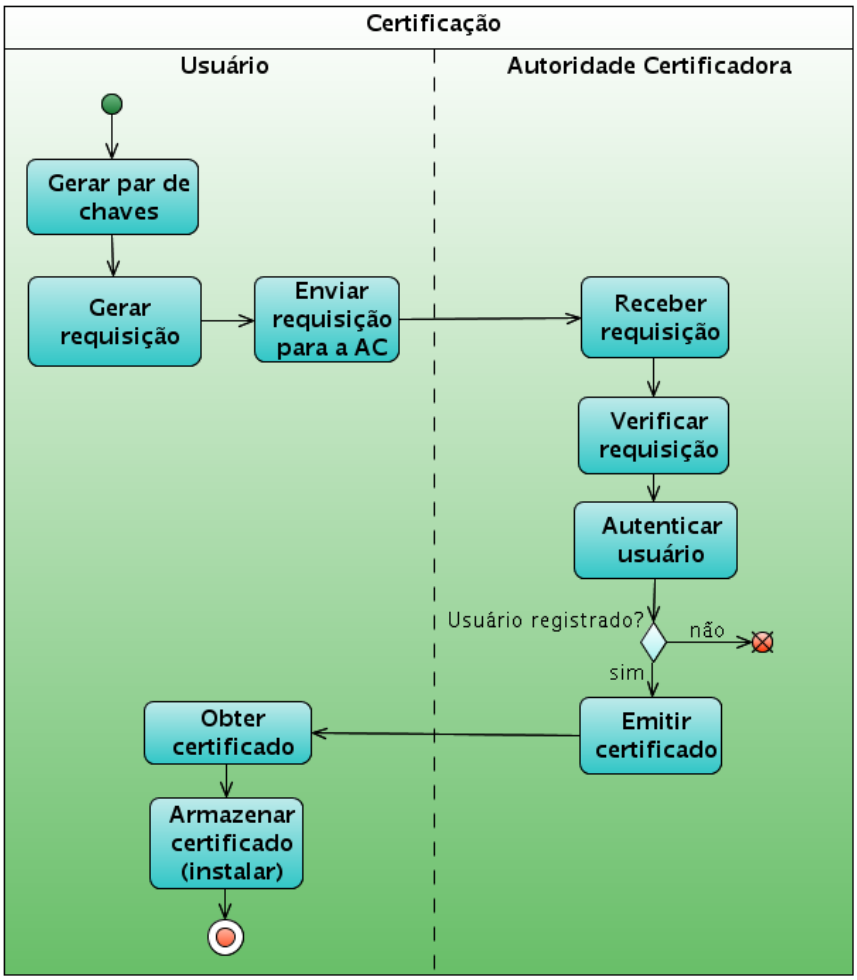

Figura 4.2: Processo de certificação

A requisição gerada é enviada para a autoridade certificadora. Após receber a requisição, a autoridade certificadora comprova a sua integridade e a auntenticidade do usuário, verificando a assinatura desta com a chave pública do usuário. Com a comprovação da identidade do usuário, o certificado é emitido pela autoridade.

Sabendo do sucesso da emissão de seu certificado, o usuário busca o seu certificado digital e o instala em seu repositório, completando o processo.

No escopo deste trabalho, assume-se que o processo de registro já tenha sido bem sucedido e portanto, o usuário já tenha sido registrado. Assume-se isto pelo fato do processo de registro ser difícil de se implementar em dispositivos móveis (pelas limitações de interface gráfica), além de ser dependente de práticas e políticas onde nem sempre uma solução on line se aplica.

Portanto, na atividade onde a autoridade certificadora autentica o usuário, utiliza-se um login e senha, sendo estes cadastrados e fornecidos ao usuário como confirmação do processo

de registro.

Apesar de existirem outras maneiras de se implementar o processo de certificação, por sua simplicidade, este foi escolhido como base para a identificação dos requisitos e desenvolvimento da ferramenta.

4.2 Requisitos

Após a análise dos processos de negócio, identificou-se que a ferramenta deve implementar as atividades do usuário no processo de certificação. Partindo desse pressuposto, identificou-se os requisitos da ferramenta, apresentados nesta seção.

Para facilitar a identificação dos requisitos funcionais da ferramenta, foi feito um levantamento dos casos de uso que a ferramenta deve suportar. A Figura 4.3 ilustra os casos de uso levantados.

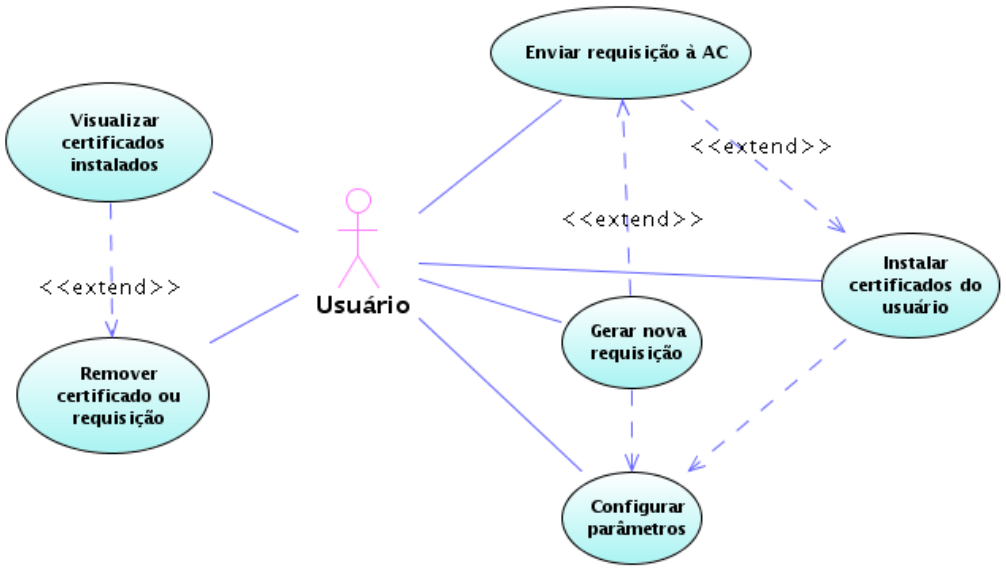

Figura 4.3: Casos de uso

No caso de uso “Gerar nova requisição”, o usuário informa o nome do titular do certificado e solicita a geração da requisição. A ferramenta, por sua vez, utilizando o smart card, gera um novo par de chaves e assina uma nova requisição. O usuário confirma a operação e informa a senha (PIN) para acesso ao smart card. Por fim, a aplicação confirma o sucesso da operação e questiona ao usuário o seu desejo de enviar a requisição para a autoridade certificadora, o que caracteriza a relação de extensão entre os casos de uso “Gerar nova requisição” e “Enviar requisição para AC”.

Se o usuário confirmar o envio, o caso de uso “Enviar requisição para AC” é executado. O usuário informa seu login e senha, obtidos no processo de registro. A ferramenta se comunica com o sistema da autoridade certificadora enviando a requisição do usuário. Após o envio, caso o certificado tenha sido emitido, a ferramenta instala o certificado e informa o sucesso ou fracasso da operação

O caso de uso “Visualizar certificados instalados” permite ao usuário a inspeção dos certificados presentes no smart card. A ferramenta apresenta uma lista com o nome do titular e o estado do certificado (Requisição pendente ou Certificado instalado). Partindo desta lista, o usuário pode visualizar as demais informações contidas no certificado. Pode também solicitar a sua exclusão, executando o caso de uso “Remover certificado ou requisição”, caracterizando outra relação de extensão representada no diagrama da Figura 4.3.

O caso de uso “Instalar certificados do usuário” é utilizado quando existe uma requisição no smart-card, o certificado desta requisição já tenha sido emitido pela autoridade certificadora, mas não tenha sido instalado ainda no smart card. Sendo assim, o usuário solicita a instalação do seu certificado digital, informando qual a requisição que foi utilizada para pedi-lo. Utilizando a URL configurada, a ferramenta faz download do certificado e o instala no repositório do smart card.

Nota-se que os casos de uso “Gerar nova requisição”, “Enviar requisição à AC” e “Instalar certificados” apresentam relação de dependência com o caso de uso “Configurar parâmetros”. Estas relações são derivadas da necessidade dos valores configurados através deste caso de uso para os seguintes parâmetros: tamanho da chave, URL de instalação de certificados, URL de envio de requisições.

Outros casos de uso foram cogitados, por exemplo, um caso de uso “Exportar certificado” poderia compor a aplicação, porém, como os certificados são armazenados no smart card, a exportação muitas vezes nem é permitida. Além disso, as demais funcionalidades de gerência de certificados, como “Instalar certificado de autoridade certificadora” e “Importar certificado de arquivo”, normalmente já são implementadas pelo sistema operacional do dispositivo móvel, portanto não necessitam estar presentes em uma ferramenta específica de obtenção de certificados digitais.

Ao visualizar os casos de uso envolvidos no escopo da aplicação, é possível identificar que os seguintes requisitos deverão ser cumpridos pela ferramenta:

- Geração de pares de chaves;
- Geração de requisições;

- Instalação de certificados digitais no smart card;
- Remoção de certificados digitais e requisições do smart card;
- Envio de requisição a uma autoridade certificadora;
- Obtenção de certificado do usuário de uma autoridade certificadora.

Tendo em vista estes requisitos, foram escolhidas algumas ferramentas para facilitar o desenvolvimento. A próxima seção apresenta tais ferramentas, justificando a escolha de cada uma delas.

4.3 Ferramentas

Nesta seção, serão descritas as ferramentas utilizadas e as justificativas para seu uso, dando subsídio para o entendimento da descrição da ferramenta desenvolvida presente na próxima seção.

4.3.1 Componentes Java ME

Como visto no capítulo anterior, a plataforma Java ME divide-se em configurações, perfis e pacotes opcionais. A lista a seguir enumera a configuração, o perfil e os pacotes opcionais utilizados para o desenvolvimento da ferramenta.

- Configuração: CLDC 1.1;
- Perfil: MIDP 2.1;
- Pacotes opcionais: SATSA 1.0;

A escolha da configuração CLDC ao invés da CDC, se justifica pelo fato desta configuração abordar os dispositivos que mais têm necessidade de uso de certificados digitais, além de terem uma maior disponibilidade de uso de smart cards para elevar o nível de segurança das aplicações, como visto no Capítulo 3.

O perfil MIDP 2.1 foi escolhido por prover facilidades para o desenvolvimento de aplicações para a configuração CLDC, como visto no capítulo 3, com destaque para o suporte ao protocolo HTTPS.

Os pacotes opcionais definidos pela SATSA 1.0, foram escolhidos para prover as funcionalidades de criptografia e acesso à smart cards. Estes pacotes não apresentam um número elevado de algoritmos e operações disponíveis como as outras bibliotecas analisadas no Capítulo 3. Porém, apesar disto, estes pacotes opcionais apresentam as seguintes vantagens que justificam a sua escolha:

- facilidade de uso;
- padronização através do JCP, provendo suporte nativo de alguns dispositivos;
- acesso a smart cards;
- classes para a geração de par de chaves, criação de requisições e remoção de certificados digitais;

Dessa maneira, os pacotes opcionais definidos pela SATSA, mesmo não sendo tão poderosos quanto a biblioteca Bouncy Castle ou a IAIK, conseguem suprir todas as necessidades da ferramenta, sendo a escolha ideal para facilitar o desenvolvimento.

Justificada as escolhas dos componentes Java ME utilizados, é necessário apresentar as ferramentas utilizadas no ambiente de desenvolvimento. A próxima seção trata deste assunto.

4.3.2 Ambiente de desenvolvimento

Para o desenvolvimento da interface gráfica da aplicação, foi utilizado como ambiente de desenvolvimento a ferramenta Netbean Mobility Pack /citeNetbeansMobilityPack. Esta ferramenta facilita muito esta tarefa ao permitir a confecção de interfaces visualmente, através de telas, transições e componentes gráficos.

Para a execução da aplicação, utilizou-se o Sun Java Wireless Toolkit for CLDC 2.5.1 (SUN MICROSYSTEMS, 2007), uma coleção de ferramentas que incluem emuladores, ferramentas de análise de desempenho, documentação e exemplos.

Dentre os emuladores, foi utilizado o emulador de dispositivo móvel fornecido pelo Sun Java Wireless Toolkit, que desempenha o papel de um telefone celular. O aparelho emulado apresenta a capacidade para utilização dos pacotes definidos pela SATSA e das demais APIs da plataforma Java ME.

Para emular a presença de um smart card, foi utilizado o emulador cref também disponível no Sun Java Wireless Toolkit.

Além do ambiente de desenvolvimento, um sistema de autoridade certificadora é necessário para o funcionamento da ferramenta. O sistema utilizado é apresentado na próxima seção.

4.3.3 Sistema da autoridade certificadora

Além das ferramentas de desenvolvimento, utilizou-se o projeto EJBCA (2007) como implementação de uma autoridade certificadora. Este projeto é baseado na tecnologia Java EE e já está consolidado, situando-se na versão 3.4.2 (utilizada neste trabalho). Além da maturidade da versão, apesar de ser um projeto de código fonte aberto, existe por trás do projeto a empresa sueca PrimeKey Solutions que mantém e presta serviços de treinamento e consultoria sobre este produto.

Dentre os protocolos aceitos, a EJBCA fornece a opção de recebimento de requisições no formato PKCS#10 através do protocolo HTTP ou HTTPS. Para a emissão do certificado, antes do envio da requisição, a entidade final já deve ter sido devidamente registrada (como descrito na analise de processos do negócio), portanto já deve possuir um login e senha. Dessa maneira, a EJBCA emite e devolve o certificado digital logo após receber a requisição e autenticar o usuário, tornando o processo mais rápido.

4.4 Descrição da ferramenta

Nesta seção será explicada a aplicação desenvolvida para atender aos requisitos identificados, abrangendo sua arquitetura, projeto de classes e projeto de interface gráfica.

4.4.1 Arquitetura

Nesta seção será descrita a arquitetura da ferramenta desenvolvida. Serão descritos os elementos de hardware e as tecnologias utilizadas na solução. A Figura 4.4 ilustra a arquitetura da ferramenta.

Devido à combinação de componentes Java ME utilizados, descrita na seção sobre as ferramentas utilizadas, a ferramenta de obtenção de certificados é uma aplicação MIDP, também chamada de MIDlet. A aplicação MIDP roda sobre a configuração CLDC no hardware do dispositivo, como mostra a Figura 4.4.

O smart card não está totalmente fora do dispositivo, pois muitas vezes o próprio dispositivo é um leitor de smart cards, como por exemplo os telefones celulares GSM.

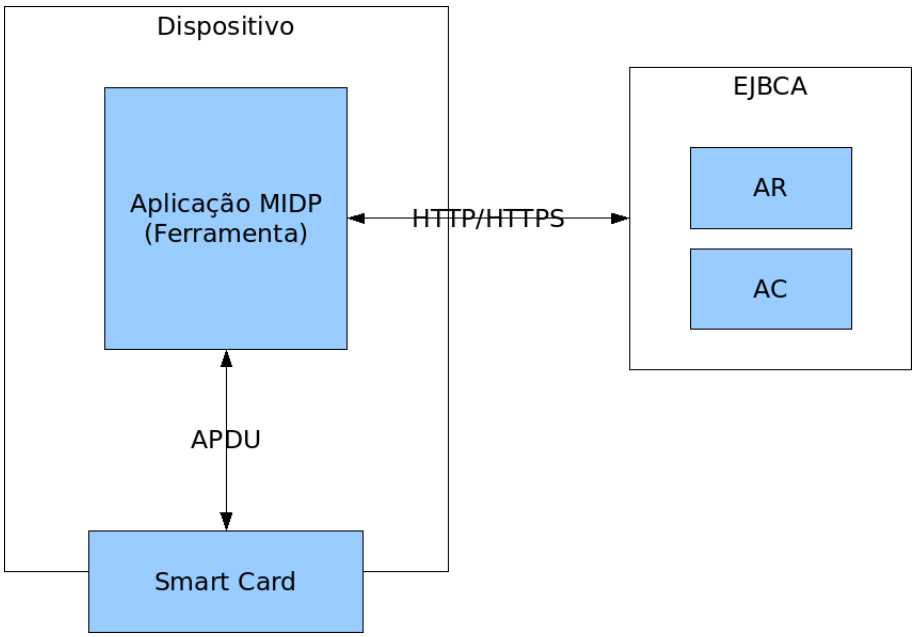

Figura 4.4: Arquitetura da ferramenta

No lado do servidor, são ilustrados os serviços das autoridades certificadoras e de registro, disponibilizados pela EJBCA.

Ainda nota-se na Figura 4.4 que existem dois canais de comunicação partindo da aplicação MIDP: um canal mostra a comunicação com o smart card e o outro com a autoridade certificadora da EJBCA.

A comunicação com o smart card é feita através de comandos APDU, realizados de maneira transparente à aplicação pelos pacotes opcionais da SATSA. Do ponto de vista da segurança, este canal é pouco vulnerável, pois a comunicação só acontece internamente ao dispositivo.

Já o outro canal de comunicação, o que permite a troca de informações entre a aplicação MIDP e a EJBCA, é mais crítico no ponto de vista da segurança. Este canal é aberto à internet e portanto é muito mais passível de ataques. Nota-se que a figura apresenta dois protocolos de comunicação neste canal, o HTTP e o HTTPS. O primeiro é utilizado quando as informações trafegadas não requerem sigilo, por exemplo quando o usuário solicita a instalação de seu certificado. O protocolo HTTPS pode ser utilizado quando há a necessidade de proteger as informações, como por exemplo quando o usuário envia seu login e senha para autenticação.

As próximas subseções descreverão a execução do processo de obtenção de certificados digitais implementado pela ferramenta. Cada subseção apresenta a execução de um caso de uso.

4.4.2 Gerar nova requisição

Em relação à interação com o usuário, o caso de uso “Gerar nova requisição” envolve a entrada do nome do titular do certificado digital, a solicitação da geração da requisição e a entrada de parâmetros necessários para acessar o smart card (PIN), como pode-se notar na Figura 4.5.

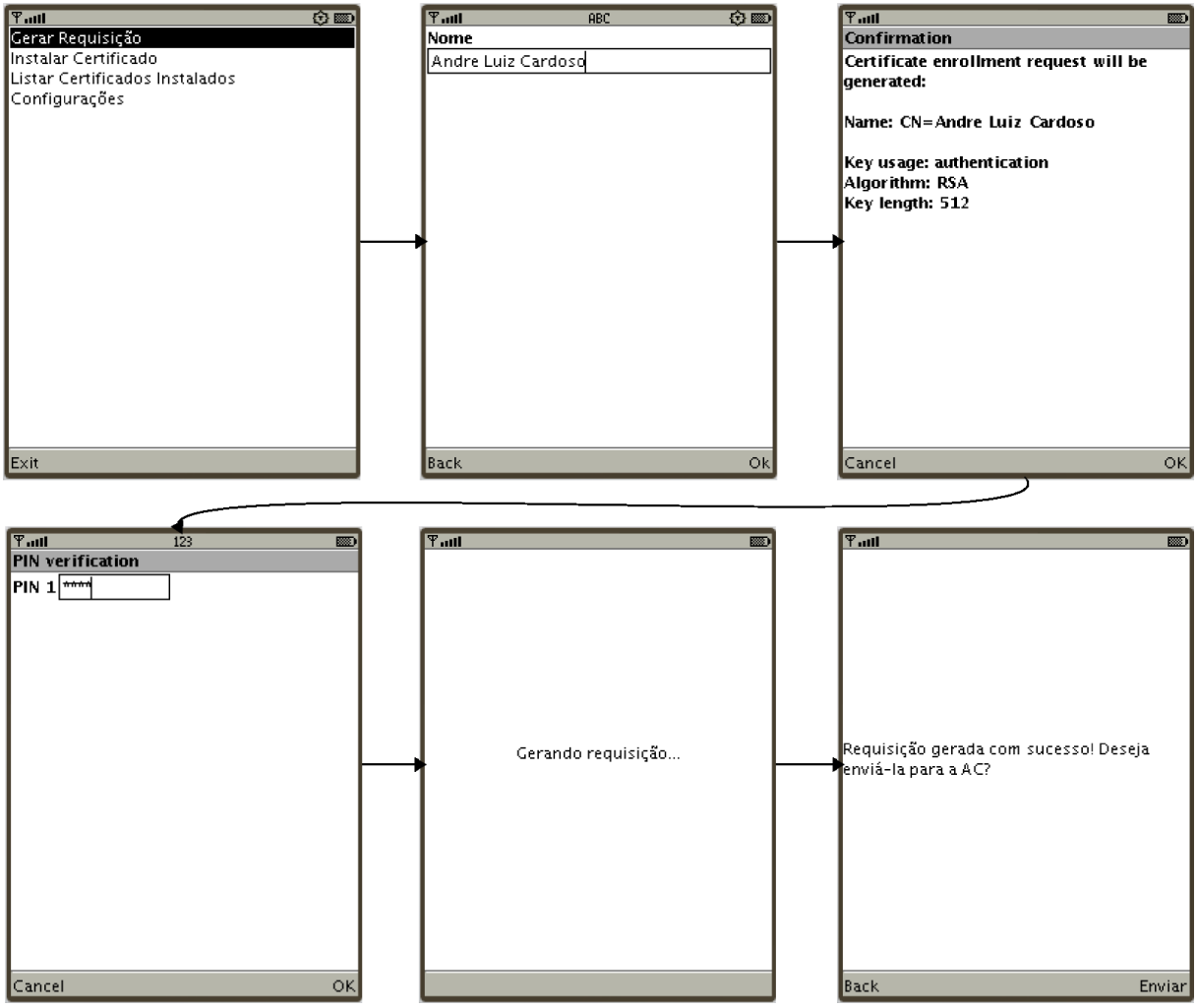

Figura 4.5: Fluxo de telas para a operação de gerar requisição

Nota-se na Figura 4.5 duas telas fornecidas pela implementação da SATSA. A terceira tela que mostra um resumo da operação de geração de requisição, solicitando a confirmação do usuário. E ainda a quarta tela que solicita a entrada do código de acesso ao smart card (PIN). O framework ainda provê uma tela que solicita a inserção do smart card, caso este não esteja presente.

Após a confirmação do usuário na terceira tela da Figura 4.5, a ferramenta acessa o smart card utilizando o PIN informado para gerar um novo par de chaves (no caso de 512 bits), gerar

uma nova requisição e assiná-la utilizando a chave privada.

A requisição é gerada no formato PKCS#10, como mostra a Figura 4.6, que ilustra a estrutura de uma requisição gerada com a ferramenta rodando nos emuladores do ambiente de desenvolvimento.

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=Andre Luiz Cardoso
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:b4:05:c4:0e:08:15:0e:a5:86:84:b2:03:00:2c:
                    cb:d7:23:85:c9:c1:0b:74:cb:20:10:b1:25:8e:4b:
                    38:d4:72:da:2c:2c:64:0e:e9:2f:b2:d6:74:2e:02:
                    77:69:74:9f:a9:98:e7:7a:4f:9e:d4:06:62:0c:33:
                    6e:80:3a:3f:e3
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Key Usage: critical
                Digital Signature
    Signature Algorithm: sha1WithRSAEncryption
        56:2c:d2:0d:f5:61:29:ec:7c:45:e7:69:33:f4:78:36:3f:cb:
        6d:52:c3:4e:28:cb:4f:64:d9:eb:8b:80:8e:71:c1:35:75:c7:
        f2:4a:e8:27:71:0b:e0:5d:7b:df:5a:04:f9:47:10:fa:cf:1a:
        33:7c:d8:7d:4d:8c:34:00:a8:57
```

Figura 4.6: Uma requisição no formato PKCS#10 gerada com a aplicação

Nota-se que esta requisição contém o nome do titular do certificado informado na segunda tela do processo. É possível notar os campos da requisição com os seguintes valores o nome do titular (Andre Luiz Cardoso), a chave pública de 512 bits, o uso da chave (assinatura digital) e por fim, a assinatura da requisição gerada utilizando o algoritmo RSA em conjunto com o hash SHA-1.

O par de chaves gerado fica armazenado no smart card. A requisição é assinada utilizando esse par de chaves, por isso, as telas do framework SATSA aparecem durante o processo.

Em relação ao processo de negócio implementado, este caso de uso compreende as duas primeiras atividades do usuário no processo de certificação (“Gerar par de chaves” e “Gerar requisição”) , como ilustra a Figura 4.2.

Após a conclusão da geração da requisição, a ferramenta questiona o usuário se deseja enviar a requisição para a autoridade certificadora configurada. Dessa maneira, o processo de obtenção de certificados digital (certificação) pode prosseguir. A próxima seção descreve o caso de uso “Enviar requisição à autoridade certificadora”, demonstrando como o restante do processo de certificação foi implementado pela ferramenta desenvolvida.

4.4.3 Enviar requisição à autoridade ceritificadora

Após a geração da requisição, esta é enviada para o endereço da autoridade certificadora configurado na ferramenta. O fluxo de telas para esta operação pode ser visualizado na Figura 4.7.

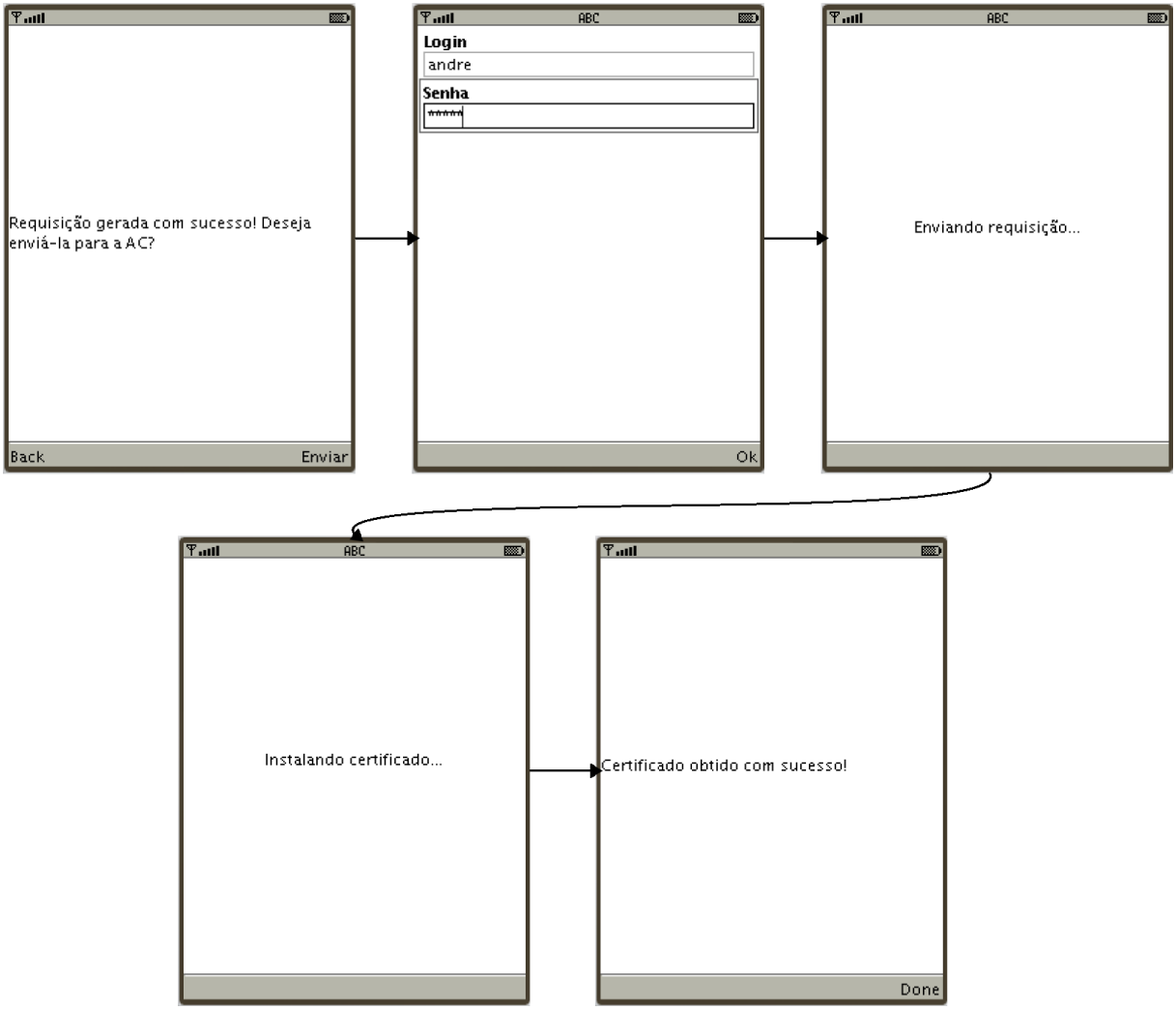

Figura 4.7: Fluxo de telas para a operação de envio de requisição para AC

A primeira tela deste caso de uso (ilustrada na Figura 4.7), é a última tela da implementação do caso de uso “Gerar nova requisição”. Isto indica que é mantida a seqüência de atividades descrita no processo de certificação (Figura 4.2).

Na segunda tela, o usuário informa o seu login e senha obtidos como confirmação do processo de registro, como visto na Figura 4.1. Estas informações são utilizadas para autenticar o usuário junto à EJBCA, onde este foi previamente cadastrado.

Nesta etapa é onde o protocolo de comunicação com a autoridade certificadora é executado. No protocolo implementado, a ferramenta empacota a requisição, o login e a senha do usuário

em uma operação POST sobre um canal seguro (HTTPS). O uso de um canal de comunicação seguro é muito importante nesta situação, pois a informação de autenticação do usuário (login e senha) tem que ser transmitida à autoridade certificadora. Ao invés do protocolo HTTPS com informações de login e senha, poderia ser utilizado um protocolo desafio-resposta para autenticar o usuário, porém a EJBCA não disponibiliza um protocolo deste tipo, sendo portanto o HTTPS a única opção segura.

Após o envio da requisição, a EJBCA verifica a autenticidade da requisição e a identidade do usuário. Caso consiga autenticá-lo, a autoridade certificadora emite o certificado e o devolve na própria sessão HTTP iniciada pela ferramenta. Dessa maneira, o usuário não necessita buscar o certificado em um momento posterior, o que torna o processo mais ágil. A penúltima tela da Figura 4.7, dá uma resposta da ferramenta ao usuário enquanto esta instala o certificado recebido da EJBCA. O final do processo é informado na última tela.

A Figura 4.8 apresenta um certificado digital obtido com a ferramenta. Este certificado é resultado da requisição apresentada na Figura 4.6.

Após a execução deste caso de uso e do caso de uso anterior, o usuário terá executado a principal funcionalidade da ferramenta, ou seja, terá obtido um novo certificado digital de maneira on line, instalando-o no seu smart card.

As demais funcionalidades da ferramenta estão descritas nas próximas subseções.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5d:56:52:1d:7b:36:27:0c
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=AC Micro Edition Intermediaria, OU=andrel, O=UFSC, C=Brasil
        Validity
            Not Before: Jun 10 23:34:59 2007 GMT
            Not After : Jun  9 23:44:59 2009 GMT
        Subject: CN=Andre Luiz Cardoso
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:b4:05:c4:0e:08:15:0e:a5:86:84:b2:03:00:2c:
                    cb:d7:23:85:c9:c1:0b:74:cb:20:10:b1:25:8e:4b:
                    38:d4:72:da:2c:2c:64:0e:e9:2f:b2:d6:74:2e:02:
                    77:69:74:9f:a9:98:e7:7a:4f:9e:d4:06:62:0c:33:
                    6e:80:3a:3f:e3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication,
                E-mail Protection, IPSec End System, IPSec User
            X509v3 Subject Key Identifier:
                AB:B1:CF:A4:7A:AA:61:98:65:D0:0B:0D:B8:FC:2A:97:42:A4:00:8D
            X509v3 Authority Key Identifier:
                keyid:39:26:34:67:70:EE:AB:C4:15:28:57:96:9D:45:16:99:22:FE:DB:B6

    Signature Algorithm: sha1WithRSAEncryption
        84:eb:21:3c:cb:5b:97:b8:33:e6:07:04:eb:55:2d:87:b0:75:
        e2:bd:9d:9b:ed:62:1a:c8:6e:6d:ee:f9:86:93:fc:27:0b:45:
        7f:b3:a3:1e:65:42:4a:6f:1c:1e:0d:8c:2e:f0:4e:6b:1f:05:
        70:27:8a:f3:2d:8d:3b:fc:77:c2:84:71:5f:08:fd:d0:e1:d2:
        1b:59:9c:8e:01:8b:82:f0:fc:c3:c4:16:4a:23:27:f1:5f:2e:
        d7:c2:8b:ec:19:89:40:3e:cd:06:9d:eb:d5:a2:2e:af:8c:af:
        11:49:80:62:6e:b6:91:04:91:bc:85:ee:e6:86:60:77:0c:93:
        0b:b5:80:60:87:55:74:36:cd:e9:41:4c:77:c9:0e:ec:bb:ff:
        be:1d:8e:ee:5e:e1:82:f6:72:89:8f:19:8a:b0:5c:d0:a9:37:
        1f:a3:a0:0d:25:76:91:ad:96:a3:85:7b:bd:75:1e:ca:f5:ad:
        f8:09:67:aa:e6:0a:d7:42:12:ae:4b:06:ec:3d:1d:d5:50:08:
        8c:c4:90:f9:a9:e8:4b:82:9a:5f:6b:90:d1:a7:7e:ba:56:61:
        92:87:70:66:4b:85:d5:58:b1:ea:1f:d0:e4:7e:ad:b1:09:76:
        a8:ad:78:9c:7a:f0:ca:77:71:07:53:56:a7:bb:33:86:52:f3:
        44:79:f5:cb
```

Figura 4.8: Certificado digital obtido com a ferramenta

4.4.4 Visualizar certificados instalados

Este caso de uso permite que o usuário saiba quais são os certificados instalados ou requisições pendentes que estão presentes no seu smart card.

A Figura 4.9 ilustra a seqüência de telas percorridas em uma execução deste caso de uso.

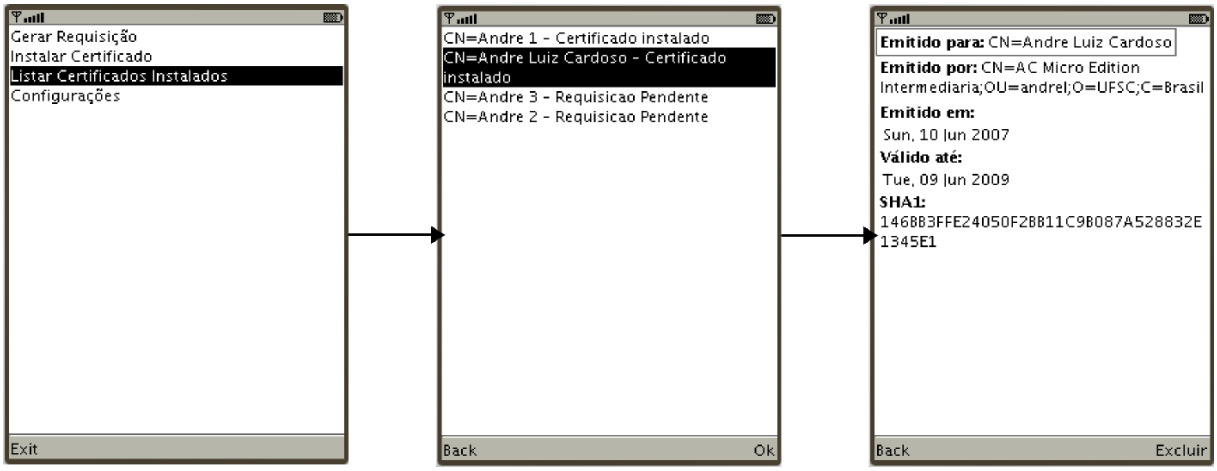

Figura 4.9: Fluxo de telas para a visualização dos certificados

Após o usuário solicitar a visualização dos certificados (indicada na primeira tela da Figura 4.9), a ferramenta exibe uma lista dos certificados digitais instalados e requisições pendentes que existem no smart card. A lista contém somente as informações do sujeito do certificado, seguidas pela situação (Certificado Instalado ou Requisição Pendente).

A partir desta lista, o usuário pode visualizar detalhes de cada certificado, como exemplifica a terceira tela da Figura 4.9. Nesta tela, são exibidas as seguintes informações:

- "Emitido para:" - apresenta as informações do sujeito do certificado (quem o requisitou e possui). No exemplo apresentado, estas informações apenas contém o nome do titular do certificado;
- "Emitido por:" - apresenta as informações da autoridade certificadora que emitiu o certificado. No exemplo apresentado, foi criada uma autoridade certificadora de teste chamada "AC Micro Edition Intermediaria";
- "Emitido em:" - apresenta a data em que o certificado foi emitido, esta data indica o ínicio da validade do certificado;
- "Válido até:" - apresenta a data em que o certificado deixa de valer;
- "SHA1:" - apresenta a impressão digital ou hash SHA-1 do certificado.

A partir da visualização dos detalhes do certificado, é possível excluí-lo do smart card. Dessa maneira, o caso de uso "Remover certificado ou requisição" é executado para exclusão do certificado ou requisição. A próxima subseção apresenta a execução deste caso de uso.

4.4.5 Remover certificado ou requisição

Os smart cards apresentam uma limitação em relação à sua capacidade de armazenamento. Para lidar com essa limitação, é interessante que a ferramenta apresente uma funcionalidade para remoção de certificados ou requisições. Além disso, pode ser necessário remover certificados revogados ou expirados.

Esta funcionalidade é disponibilizada pela ferramenta, a partir da visualização dos detalhes do certificado a ser excluído. A Figura 4.10 apresenta o fluxo de telas da remoção de um certificado.

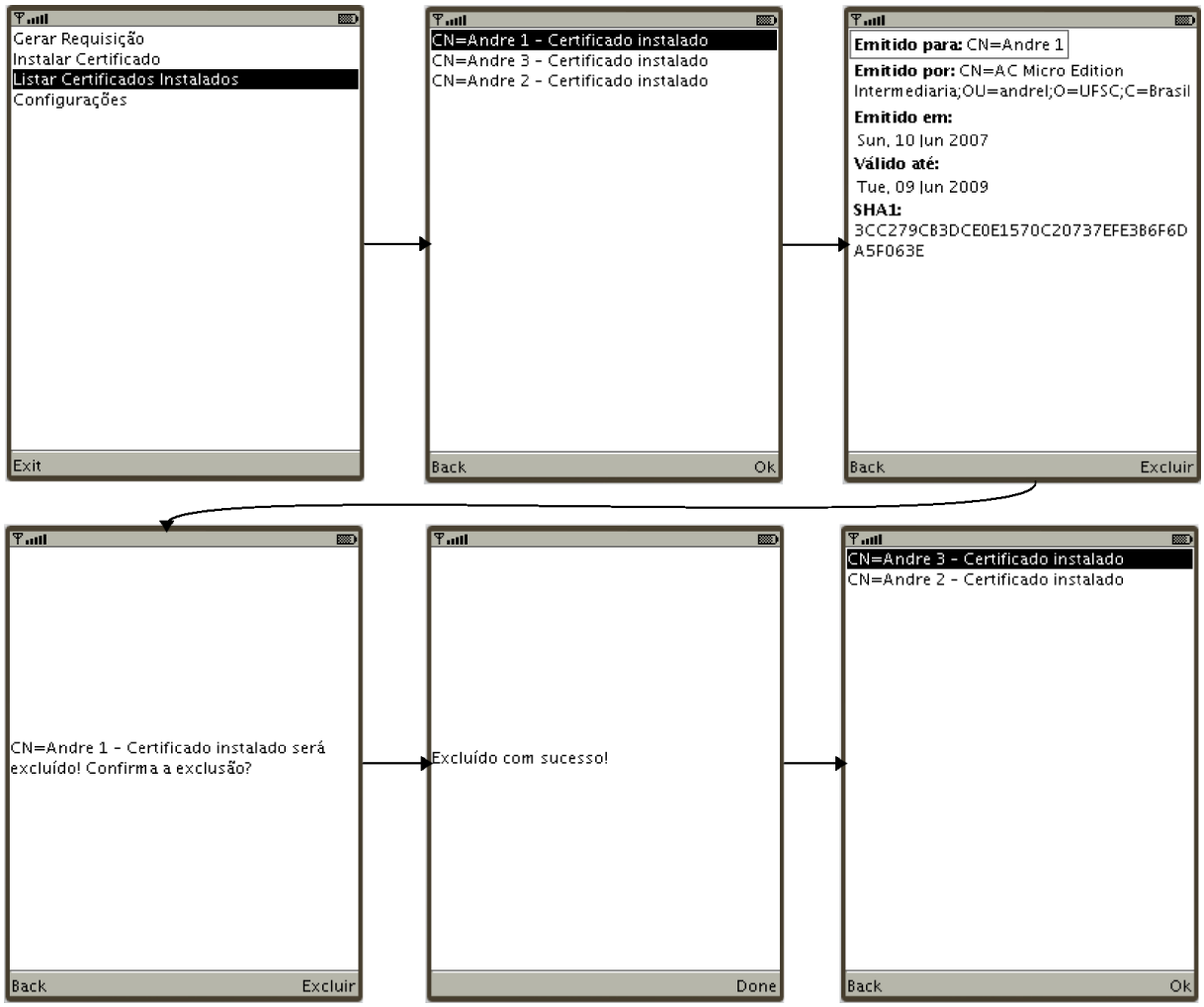

Figura 4.10: Fluxo de telas para a remoção de certificados ou requisições

Antes de concretizar a operação de exclusão do certificado no smart card, a ferramenta solicita a confirmação do usuário, como é ilustrado na quarta tela da Figura 4.10.

Após a confirmação da exclusão, a operação é realizada e a lista de certificados volta a ser exibida.

4.4.6 Instalar certificados do usuário

No caso de uso “Enviar requisição à autoridade certificadora”, o protocolo implementado pela ferramenta, recebe o certificado emitido e o instala no smart card. Caso não seja possível obter o certificado logo após o envio da requisição, a ferramenta possibilita a obtenção e instalação do certificado emitido, em um momento posterior ao envio da requisição. A execução desta funcionalidade é ilustrada na Figura 4.11.

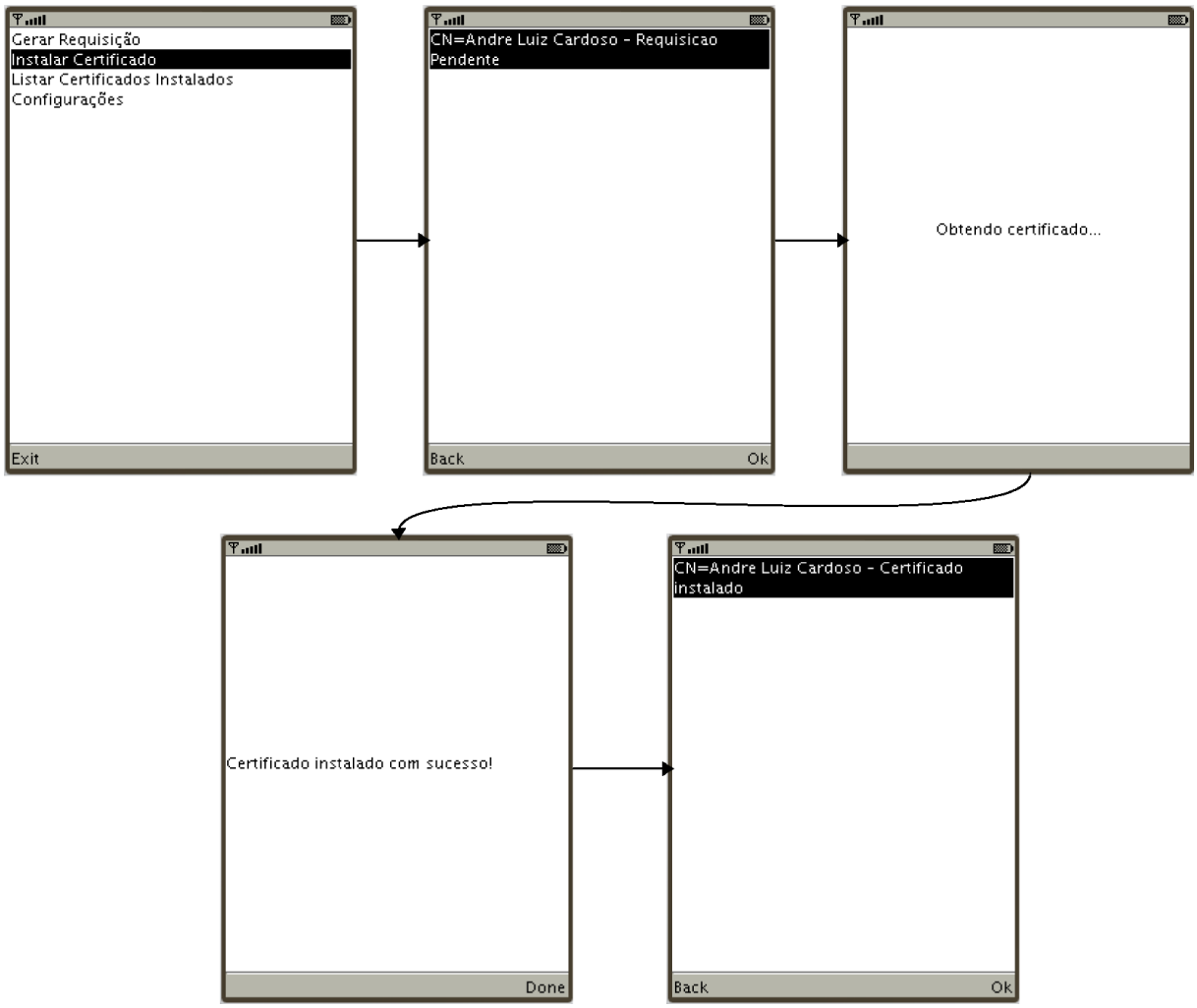

Figura 4.11: Fluxo de telas para a instalação de certificados do usuário

Na primeira tela, o usuário solicita a instalação de certificado, a ferramenta então, apresenta uma lista contendo as requisições cujos certificados ainda não foram instalados (Requisição Pendente), como ilustra a segunda tela da Figura 4.11. Após selecionar uma dessas requisições (no exemplo só existe uma), a ferramenta utiliza o endereço configurado e os dados da requisição para obter o certificado digital junto à autoridade certificadora, como ilustra a terceira tela. Após baixar o certificado, a ferramenta o instala junto ao smart card e apresenta uma mensagem informando o sucesso da operação. O certificado instalado pode ser verificado na lista exibida

na última tela da Figura 4.11.

A próxima subseção apresenta a execução do caso de uso “Configurar parâmetros”, onde o usuário pode configurar parâmetros utilizados em outros casos de uso.

4.4.7 Configurar parâmetros

Para gerar o par de chaves no caso de uso “Gerar nova requisição”, a ferramenta precisa conhecer o tamanho da chave a ser gerada.

No caso de uso “Enviar a requisição à autoridade certificadora”, a ferramenta precisa conhecer o endereço (URL) no qual a autoridade certificadora recebe as requisições.

No caso de uso “Instalar certificados do usuário”, a ferramenta precisa conhecer o endereço no qual a autoridade certificadora disponibiliza o serviço de obtenção de certificados.

Para configurar estes parâmetros, o usuário utiliza a operação ilustrada pela Figura 4.12.

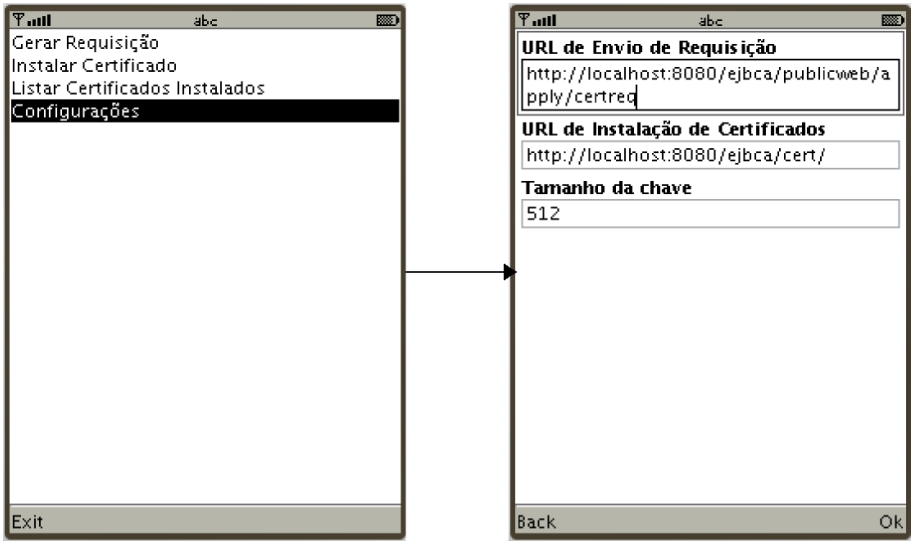

Figura 4.12: Fluxo de telas para a configuração de parâmetros

Esta é a funcionalidade mais simples da ferramenta, onde em uma só tela, o usuário configura os três parâmetros utilizados em outros casos de uso.

4.5 Resultados

O maior resultado deste trabalho é a ferramenta para obtenção on line de certificados digitais para o uso de smart cards em aplicações Java ME. A integração da aplicação com uma autoridade certificadora (EJBCA) foi possível e funcional. O armazenamento das chaves e a

realização de operações critptográficas em smart cards também foi possível e de fácil implementação, devido à escolha do framework SATSA.

A aplicação só foi executada em ambiente de desenvolvimento e portanto o seu comportamento em dispositivos reais não foi comprovado. Porém, por ter utilizado componentes padronizados e emuladores confiáveis, pode-se perceber que a implantação da ferramenta em dispositivos reais não será uma tarefa difícil.

Ainda assim, com o seu desenvolvimento, provou-se que o uso de certificados digitais em dispositivos móveis é viável, se o dispositivo tiver recursos computacionais suficientes para executar operações criptográficas.

4.6 Conclusão

Pode-se concluir que a escolha das ferramentas corretas possibilitou o desenvolvimento da ferramenta. A abstração, por parte dos frameworks e bibliotecas utilizados, sobre a complexidade de certas operações como o acesso a smart cards e a geração de requisições e chaves, facilitou muito a implementação dos casos de uso que necessitavam realizar tais operações.

Conclui-se ainda que a ferramenta desenvolvida pode ser utilizada por usuários avançados, que desejam que suas aplicações Java ME possam utilizar certificados digitais instalados em seus smart cards.

5 *Conclusão*

Uma aplicação de obtenção on line de certificados digitais para uso de smart cards em aplicações Java ME como a proposta por este trabalho tem a sua importância em servir de base para prover níveis de segurança mais elevados às aplicações Java ME.

A aplicação desenvolvida facilita a instalação de certificados digitais em dispositivos móveis, desta maneira possibilita que outras aplicações utilizem os certificados instalados para resolver problemas de segurança.

O fato da ferramenta possibilitar o uso de smart cards para armazenamento de certificados digitais e chaves é uma interessante contribuição para que o uso destas tecnologias seja mais difundido. A ferramenta pode ser utilizada por usuários avançados, que desejam que suas aplicações Java ME utilizem certificados digitais instalados em seus smart cards.

Conclui-se que a aplicação desenvolvida é um passo a mais para a popularização da tecnologia de certificados digitais. Autoridades certificadoras comerciais, governamentais ou corporativas poderão disponibilizar a seus clientes uma aplicação como a desenvolvida, agregando ainda mais usuários a seus serviços.

A tecnologia de certificados digitais está em crescimento e é uma solução viável para muitos problemas de segurança da informação nas aplicações atuais. A união desta tecnologia com a portabilidade e a popularidade dos dispositivos móveis que suportam Java ME, pode trazer uma série de benefícios e possibilitar o desenvolvimento de aplicações seguras em dispositivos móveis.

5.1 Trabalhos futuros

Uma série de trabalhos futuros podem ser executados a partir da ferramenta desenvolvida. Os trabalhos que derivam mais diretamente são aplicações típicas de dispositivos móveis que necessitem da tecnologia de certificados digitais.

Uma destas aplicações é a conjunto com a súmula virtual proposta por George Ramos Martins e Rodolpho Luna de Moura (2006). Dessa maneira, sugere-se que um trabalho futuro integre ambas as aplicações em um dispositivo, resolvendo os problemas de segurança descritos em tal trabalho.

Outra proposta de aplicação que utiliza a tecnologia de certificados digitais é a automatização do sistema de multas, utilizando os certificados digitais para assinatura das multas por parte de quem as aplica, em uma aplicação de dispositivos móveis.

Assim como estas duas aplicações, os certificados digitais podem ser utilizados em muitas outras, sendo que em todas elas, a obtenção de certificados digitais é necessária. Portanto, a ferramenta desenvolvida vem para contribuir ao possibilitar que esta operação seja executada.

Além da integração desta aplicação com outras soluções, um trabalho mais imediato poderia tratar do teste desta aplicação em dispositivos reais, validando o funcionamento em dispositivos de diversos fabricantes.

Um trabalho futuro mais ambicioso seria a extensão da aplicação para permitir o uso do protocolo CMP definido por C. Adams S. Farrell e T. Mononen (2005) (RFC 4210). A adoção deste protocolo traria uma maior interoperabilidade com as autoridades certificadoras. Isto seria uma grande vantagem, pois a aplicação ficaria independente do sistema da autoridade certificadora.

Referências Bibliográficas

Andre N. Klingsheim; Veborn Moen; Kjell J. Hole. Challenges in securing networked j2me applications. In: **Computer**. [s.n.], 2007. v. 40, n.2, p. 24–30. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/MC.2007.49>.

BOK, L. Y. **Evolution Directions of Mobile Device and Its Factors**. [S.l.], 2006. 10 p. Disponível em: <http://www.infoedge.com/productz_type.asp?product=RO-0001>.

C. Adams S. Farrell, T.; T. Mononen. **Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)**. [S.l.], September 2005. RFC 4210. Disponível em: <http://www.faqs.org/rfcs/rfc4210.html>.

Carlisle Adams; Steve Lloyd. **Understanding PKI: Concepts, Standards, and Deployment Considerations**. 2nd. ed. [S.l.]: Addison Wesley, 2002. ISBN 0-672-32391-5.

EJBCA. **Facts that every IT-security professional needs to know about EJBCA**. [S.l.], 2007. Disponível em: <http://download.primekey.se/documents/ejbca_techinfo.pdf>.

George Ramos Martins; Rodolpho Luna de Moura. **Súmula Virtual: Modernização da Súmula de Futebol utilizando PDAs interligados com um Web Service.** 2006.

GIGUERE, E. **J2ME Optional Packages**. [S.l.], 2002. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/articles/optional/index.html>.

HOUSLEY, R. **Cryptographic Message Syntax (CMS)**. [S.l.], July 2004. RFC 3852. Disponível em: <http://www.faqs.org/rfcs/rfc3852.html>.

IAIK. **JCE-ME**. [S.l.], 2007. Disponível em: <http://jce.iaik.tugraz.at/sic/products/core_crypto_toolkits/jce_me>.

JAVA CARD FORUM. **Java Card Forum**. [S.l.], 2007. Disponível em: <http://www.javacardforum.org/>.

KNUDSEN, J. **Wireless Java Developing with J2ME**. 2nd. ed. [S.l.]: Apress, 2003. ISBN 1-59059-077-5.

M. Nystrom; B. Kaliski. **PKCS #10: Certification Request Syntax Specification**. [S.l.], November 2000. RFC 2986. Disponível em: <http://www.faqs.org/rfcs/rfc2986.html>.

MCATEER, S. **Java Versus BREW**. [S.l.], 2004. 14 p. Disponível em: <http://www.zelosgroup.com>.

ORTIZ, E. **A Survey of J2ME Today**. [S.l.], 2004. Disponível em: <http://developers.sun.com/techtopics/mobility/getstart/articles/survey/>.

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

R. L. Rivest; A. Shamir; L. M. Adelman. **A Method for Obtaining Digital Signatures and Public-key Cryptosystems**. [S.l.], 1977. 15 p. Disponível em: <http://citeseer.ist.psu.edu/rivest78method.html>.

RSA. **The Cryptographic Smart Card:A Portable, Integrated Security Platform**. [S.l.], 2001. Disponível em: <http://www.afina.com.mx/download/docs/rsa/SecurIDSmartCard.pdf>.

Russ Housley; Tim Polk. **Plannig for PKI: Best Practices Guide for Deploying Public Key Infrastructure**. [S.l.]: Wiley, 2001.

SUN MICROSYSTEMS. **Datasheet of Security and Trust Services APIs For the Java 2 Platform, Micro Edition**. [S.l.]. Disponível em: <http://java.sun.com/j2me/docs/sec_trust-177.pdf>.

SUN MICROSYSTEMS. **SATSA Developer's Guide, SATSA Reference Implementation 1.0**. [S.l.], 2004. Disponível em: <http://java.sun.com/j2me/docs/satsa-dg/index.html>.

SUN MICROSYSTEMS. **Security and Trust Services API (SATSA) for the Java 2 Platform, Micro Edition**. [S.l.], 2004. Disponível em: <http://www.jcp.org/en/jsr/detail?id=177>.

SUN MICROSYSTEMS. **CDC: Java Platform Technology for Connected Devices**. [S.l.], 2005. Disponível em: <http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf>.

SUN MICROSYSTEMS. **Mobile Information Device Profile for Java2 Micro Edition**. [S.l.], 2006. Disponível em: <http://www.jcp.org/en/jsr/detail?id=118>.

SUN MICROSYSTEMS. **Java ME Technology**. [S.l.], 2007. Disponível em: <http://java.sun.com/javame/technology>.

SUN MICROSYSTEMS. **Mobility Overview**. [S.l.], 2007. Disponível em: <http://developers.sun.com/techtopics/mobility/overview.html>.

SUN MICROSYSTEMS. **Sun Java Wireless Toolkit for CLDC**. [S.l.], 2007. Disponível em: <http://java.sun.com/products/sjwtoolkit/>.

WARD, R. **Survey of Cryptographic Smart Card Capabilities and Vulnerabilities**. [S.l.], 2001. Disponível em: <http://ece.gmu.edu/courses/ECE636/project/reports/RWard.pdf>.

Whitfield Diffie; Martin Hellman. New directions in cryptography. In: **IEEE Transactions on Information Theory**. [s.n.], 1976. IT-22, p. 644–654. Disponível em: <http://citeseer.ist.psu.edu/diffie76new.html>.

William Stallings. **Cryptography and Network Security: Principles and Practice**. 2nd. ed. [S.l.]: Prentice Hall, 1998.

YUAN, M. J. **Enterprise J2ME: Developing Mobile Java Applications**. [S.l.]: Prentice Hall PTR, 2003. ISBN 0-13-140530-6.

Anexos

Anexo A - Diagrama de classes

A Figura 5.1 ilustra as classes projetadas na forma de diagrama.

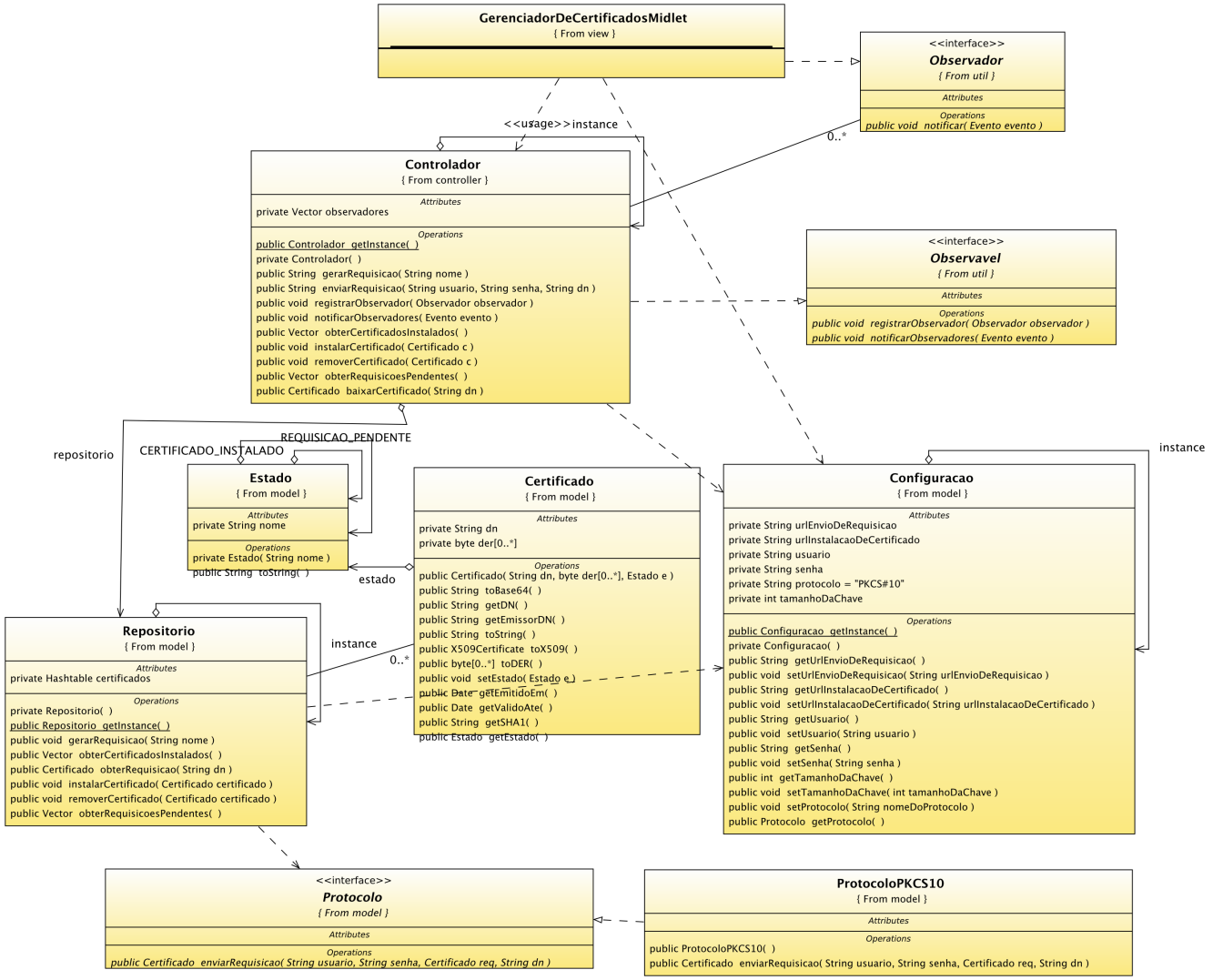

Figura 5.1: Diagrama de classes

Anexo B - Código fonte da ferramenta

Para quem for curioso e quiser visualizar como a ferramenta foi implementada, este anexo terá grande valia.

5.0.1 Pacote br.ufsc.inf.andrel.tcc.model

Neste pacote se encontram as classes referentes ao negócio, contendo portanto a maior parte da lógica da ferramenta.

Classe Certificado

```
/**
 * Classe do modelo que representa um certificado digital que pode estar em 2
 * estados (indicado pelo atributo estado) que são: Estado.CERTIFICADO_INSTALADO
 * ou Estado.REQUISICAO_PENDENTE.
 */
public class Certificado {

    private String dn;
    private byte[] der;
    private Estado estado;

    /** Creates a new instance of Certificado */
    public Certificado(String dn, byte[] der, Estado e) {
        this.dn = dn;
        this.der = der;
        this.estado = e;
    }

    public String toBase64() {
        return Base64.encode(der,0,der.length);
    }

    public String getDN() {
        return dn;
```

```
}

public String getEmissorDN() {
    return toX509().getIssuer();
}

public String toString() {
    return dn + " - " + estado;
}

public X509Certificate toX509() {
    try {
        return X509Certificate.generateCertificate(der,0,der.length);
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}

public byte[] toDER() {
    return der;
}

public void setEstado(Estado e) {
    this.estado = e;
}

public Date getEmitidoEm() {
    Date d = new Date(toX509().getNotBefore());
    return d;
}

public Date getValidoAte() {
    Date d = new Date(toX509().getNotAfter());
    return d;
```

```
    }

    public String getSHA1() {
        byte[] buffer = new byte[20];
        try {
            MessageDigest hash = MessageDigest.getInstance("SHA-1");
            hash.update(der,0,der.length);
            hash.digest(buffer,0,buffer.length);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return new String(Hex.encode(buffer)).toUpperCase();
    }

    public Estado getEstado() {
        return estado;
    }
}
```

Classe Configuracao

```
/**
 *
 * Classe do modelo que representa a configuração atual da ferramenta.
 */
public class Configuracao {

    private String urlEnvioDeRequisicao;
    private String urlInstalacaoDeCertificado;
    private String usuario;
    private String senha;
    private String protocolo = "PKCS#10";
    private int tamanhoDaChave;


    private static Configuracao instance;
```

```
public static Configuracao getInstance() {
    if(instance == null) {
        instance = new Configuracao();
    }
    return instance;
}

/** Creates a new instance of Configuracao */
private Configuracao() {
    tamanhoDaChave = 512;
    urlEnvioDeRequisicao = "http://localhost:8080/ejbca/publicweb/apply/certreq
    urlInstalacaoDeCertificado = "http://localhost:8080/ejbca/cert/";
}

public String getUrlEnvioDeRequisicao() {
    return urlEnvioDeRequisicao;
}

public void setUrlEnvioDeRequisicao(String urlEnvioDeRequisicao) {
    this.urlEnvioDeRequisicao = urlEnvioDeRequisicao;
}

public String getUrlInstalacaoDeCertificado() {
    return urlInstalacaoDeCertificado;
}

public void setUrlInstalacaoDeCertificado(String urlInstalacaoDeCertificado) {
    this.urlInstalacaoDeCertificado = urlInstalacaoDeCertificado;
}

public String getUsuario() {
    return usuario;
}
```

```
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public int getTamanhoDaChave() {
        return tamanhoDaChave;
    }

    public void setTamanhoDaChave(int tamanhoDaChave) {
        this.tamanhoDaChave = tamanhoDaChave;
    }

    public void setProtocolo(String nomeDoProtocolo) {
        this.protocolo = nomeDoProtocolo;
    }

    public Protocolo getProtocolo() {
        if(protocolo == "PKCS#10") {
            return new ProtocoloPKCS10();
        }

        return null;
    }
}
```

Classe Estado

```
/**
```

```
 *
 * Classe que simula uma Enumeration contendo os estados do certificado.
 */
public class Estado {

    public static Estado REQUISICAO_PENDENTE = new Estado("Requisicao Pendente");
    public static Estado CERTIFICADO_INSTALADO =
            new Estado("Certificado instalado");

    private String nome;

    /** Creates a new instance of Estado */
    private Estado(String nome) {
        this.nome = nome;
    }

    public String toString() {
        return nome;
    }

}
```

Interface Protocolo

```
/**
 *
 * Interface utilizada para tornar mais fácil a expansão da ferramenta para uso
 * de outros protocolos de comunicação entre a ferramenta e a
 * autoridade certificadora.
 */
public interface Protocolo {
    Certificado enviarRequisicao(String usuario, String senha, Certificado req, Str
}
```

Classe ProtocoloPKCS10

```
/**
 *
 * Implementação do protocolo de comunicação entre a ferramenta e a EJBCA.
 */
public class ProtocoloPKCS10 implements Protocolo {

    /** Creates a new instance of ProtocoloPKCS10 */
    public ProtocoloPKCS10() {
    }

    public Certificado enviarRequisicao(String usuario, String senha,
            Certificado req, String dn) {
        try {

            HttpConnection connection = (HttpConnection)Connector.open(
                    Configuracao.getInstance().getUrlEnvioDeRequisicao());
            connection.setRequestMethod(HttpConnection.POST);
            connection.setRequestProperty("Content-Type",
                    "application/x-www-form-urlencoded");
            connection.setRequestProperty("Accept","text/html," +
                    " image/gif, image/jpeg, *; q=.2, */*; q=.2");
            connection.setRequestProperty("Connection","keep-alive");

            String content = "pkcs10req=" +
                    URLEncoder.encode("-----BEGIN CERTIFICATE REQUEST-----\n" +
                    req.toBase64() + "\n-----END CERTIFICATE REQUEST-----\n") +
                    "&password=" + URLEncoder.encode(senha) +
                    "&user=" + URLEncoder.encode(usuario) +
                    "&resulttype=1&submit=Submit+Query";

            System.out.println(req.toBase64());

            DataOutputStream output = connection.openDataOutputStream();
```

```
byte[] data = content.getBytes();
connection.setRequestProperty("Content-Length",
        Integer.toString(data.length));

output.write(data);

if(connection.getResponseCode() != HttpConnection.HTTP_OK) {
    int codigo = connection.getResponseCode();
    output.close();
    connection.close();
    throw new RuntimeException("Erro " + codigo
            + " ao enviar a requisi\u00E7\u00E3o.");
}

DataInputStream input = connection.openDataInputStream();

byte[] certificado = new byte[input.available()];
input.read(certificado);
input.close();
connection.close();

String b64 = new String(certificado);

System.out.println(b64);

b64 = b64.substring("-----BEGIN CERTIFICATE-----\n".length());
b64 = b64.substring(0,b64.length()-"\n-----END CERTIFICATE-----".
        length());

StringBuffer certificadoB64 = new StringBuffer();
for (int i = 0; i < b64.length(); i++) {
    char c = b64.charAt(i);
    if(c != '\n')
        certificadoB64.append(c);
}
```

```
            return new Certificado(dn,Base64.decode(certificadoB64.toString()),
                    Estado.CERTIFICADO_INSTALADO);

        } catch (IOException ex) {
            ex.printStackTrace();
            throw new RuntimeException("Erro ao enviar a requisi\u00E7\u00E3o.");
        }
    }

}
```

Classe Repositorio

```
/**
 *
 * Esta classe representa o repositório de certificados presentes no smart card.
 */
public class Repositorio {

    private Hashtable certificados;

    private static Repositorio instance;

    /** Creates a new instance of Repositorio */
    private Repositorio() {
        certificados = new Hashtable();
    }

    public static Repositorio getInstance() {
        if(instance == null) {
            instance = new Repositorio();
        }
        return instance;
    }
```

```
public void gerarRequisicao(String nome) {
    try {
        byte[] requisicao = UserCredentialManager.generateCSR(nome,
                UserCredentialManager.ALGORITHM_RSA,
                Configuracao.getInstance().getTamanhoDaChave(),
                UserCredentialManager.KEY_USAGE_AUTHENTICATION,
                null,
                "Please, insert the security element.",
                true);
        certificados.put(nome,new Certificado(nome,
                requisicao,Estado.REQUISICAO_PENDENTE));
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(
                "Falha na gera\u00E7\u00E3o da requisi\u00E7\u00E3o.");
    }

}

public Vector obterCertificadosInstalados() {
    Vector vector = new Vector();

    Enumeration elements = certificados.elements();
    while(elements.hasMoreElements()) {
        vector.addElement(elements.nextElement());
    }

    return vector;
}

public Certificado obterRequisicao(String dn) {
    return (Certificado) certificados.get(dn);
}
```

```
public void instalarCertificado(Certificado certificado) {
    try {
        UserCredentialManager.addCredential(certificado.getDN(),
                certificado.toDER(),null);
        certificados.put(certificado.getDN(),certificado);
    } catch (UserCredentialManagerException ex) {
        ex.printStackTrace();
    }
}

public void removerCertificado(Certificado certificado) {
    try {

        UserCredentialManager.removeCredential(certificado.getDN(),
                certificado.toX509().getIssuer().getBytes(),
                null,
                "Please insert the security element.");
    } catch (UserCredentialManagerException ex) {
        ex.printStackTrace();
    }

    certificados.remove(certificado.getDN());
}

public Vector obterRequisicoesPendentes() {
    Vector vector = new Vector();

    Enumeration elements = certificados.elements();
    while(elements.hasMoreElements()) {
        Certificado c = (Certificado) elements.nextElement();
        if(c.getEstado() == Estado.REQUISICAO_PENDENTE) {
            vector.addElement(c);
        }
    }
```

```
        return vector;
    }

}
```

5.0.2 Pacote br.ufsc.inf.andrel.tcc.controller

Neste pacote se encontra a classe referente ao controle da aplicação.

Classe Controlador

```
/**
 * Classe que controla o fluxo de operações, notifica a interface gráfica com
 * as mudanças nos objetos do modelo e delega o trabalho para as classes
 * do modelo.
 **/
public class Controlador implements Observavel {

    private static Controlador instance;

    private Vector observadores;

    private Repositorio repositorio = Repositorio.getInstance();

    public static Controlador getInstance() {
        if(instance == null) {
            instance = new Controlador();
        }
        return instance;
    }

    private Controlador() {
        observadores = new Vector();
    }

    public String gerarRequisicao(String nome) {
```

```
        notificarObservadores(Evento.GERANDO_REQUISICAO);
        nome = "CN=" + nome;

        repositorio.gerarRequisicao(nome);

        notificarObservadores(Evento.REQUISICAO_GERADA);

        return nome;
    }

    public String enviarRequisicao(String usuario, String senha, String dn) {
        notificarObservadores(Evento.ENVIANDO_REQUISICAO);
        Certificado req = repositorio.obterRequisicao(dn);

        Protocolo protocolo = Configuracao.getInstance().getProtocolo();

        Certificado c = protocolo.enviarRequisicao(usuario,senha,req,dn);
        notificarObservadores(Evento.REQUISICAO_ENVIADA);

        instalarCertificado(c);

        return "OK";
    }

    public void registrarObservador(Observador observador) {
        observadores.addElement(observador);
    }

    public void notificarObservadores(Evento evento) {
        for(int i = 0; i < observadores.size(); i++) {
            ((Observador)observadores.elementAt(i)).notificar(evento);
        }
    }

    public Vector obterCertificadosInstalados() {
```

```
        return repositorio.obterCertificadosInstalados();
    }

    public void instalarCertificado(Certificado c) {
        notificarObservadores(Evento.INSTALANDO_CERTIFICADO);

        c.setEstado(Estado.CERTIFICADO_INSTALADO);
        repositorio.instalarCertificado(c);

        notificarObservadores(Evento.CERTIFICADO_INSTALADO);
    }

    public void removerCertificado(Certificado c) {
        repositorio.removerCertificado(c);
    }

    public Vector obterRequisicoesPendentes() {
        return repositorio.obterRequisicoesPendentes();
    }

    public Certificado baixarCertificado(String dn) {
        notificarObservadores(Evento.OBTENDO_CERTIFICADO);

        notificarObservadores(Evento.CERTIFICADO_OBTIDO);
        return null;
    }
}
```

5.0.3 Pacote br.ufsc.inf.andrel.tcc.util

Neste pacote se encontram as classes utilitárias utilizadas em todo o projeto.

Classe Evento

```
/**
 *
```

```
 * Classe que simula uma Enumeration contendo os eventos que devem ser avisados
 * para a interface gráfica.
 */
public class Evento {

    public static Evento GERANDO_REQUISICAO =
            new Evento("Gerando requisi\u00E7\u00E3o...");
    public static Evento REQUISICAO_GERADA =
            new Evento("Requisi\u00E7\u00E3o gerada com sucesso!");
    public static Evento ENVIANDO_REQUISICAO =
            new Evento("Enviando requisi\u00E7\u00E3o...");
    public static Evento REQUISICAO_ENVIADA =
            new Evento("Requisi\u00E7\u00E3o enviada para a AC com sucesso!");
    public static Evento INSTALANDO_CERTIFICADO =
            new Evento("Instalando certificado...");
    public static Evento CERTIFICADO_INSTALADO =
            new Evento("Certificado instalado com sucesso!");
    public static Evento OBTENDO_CERTIFICADO =
            new Evento("Obtendo certificado...");
    public static Evento CERTIFICADO_OBTIDO =
            new Evento("Certificado obtido com sucesso!");

    private String mensagem;

    private Evento(String mensagem) {
        this.mensagem = mensagem;
    }

    public String toString() {
        return mensagem;
    }

}
```

Interface Observador

```
/**
 *
 * Interface para suporte ao padrão de projeto Observador, utilizado neste
 * projeto para comunicação entre o controlador e a interface gráfica.
 */
public interface Observador {
    public void notificar(Evento evento);
}
```

Interface Observavel

```
/**
 *
 * Interface para suporte ao padrão de projeto Observador, utilizado neste
 * projeto para comunicação entre o controlador e a interface gráfica.
 */
public interface Observavel {

    void registrarObservador(Observador observador);

    void notificarObservadores(Evento evento);

}
```

Classe URLEncoder

```
/**
 * Esta classe foi obtida de um projeto livre de código aberto sem restrições.
 * Esta classe é um helper para a classes que utilizem o protocolo HTTP para
 * comunicação.
 * Esta classe fornece métodos para codificar dados no formato das URLs,
 * permitindo que sejam transmitidos sobre tal protocolo.
 */
public class URLEncoder {
```

```
private static char getHex(int val) {
    if (val < 10) return (char)('0' + val);
    else return (char)('A' + (val - 10));
}

public static String encode(String url) {

    StringBuffer returnURL = new StringBuffer();

    int size = url.length();

    for(int i = 0; i < size; i++) {

        // iterate over each character
        char ch = url.charAt(i);

        // if alphnumeric, it remains as such
        if ((ch >= '0' && ch <= '9') ||
                (ch >= 'a' && ch <= 'z') ||
                (ch >= 'A' && ch <= 'Z'))
            returnURL.append(ch);
        else {

            // non alphanumeric

            // see first if this is one of the special characters.
            int spec = special.indexOf(ch);

            if (spec >= 0) {

                // this character is not in the special chars
                // String defined later
                returnURL.append(ch);
```

```
                } else {

                    // use the hex converter for the rest of the characters

                    // first add the % character
                    returnURL.append('%');

                    // next convert the high bits
                    returnURL.append(getHex((ch & 0xF0) >> 4));

                    // and finally the low bits
                    returnURL.append(getHex(ch & 0x0F));
                }
            }
        }

        // the final encoded url
        return returnURL.toString();
    }

    private static String special = "=&:/?\".-!~*_'()";
}
```

5.0.4 Pacote br.ufsc.inf.andrel.tcc.view

Neste pacote se encontra a classe responsável pela interface gráfica da ferramenta.

Classe GerenciadorDeCertificadosMIDlet

```
/**
 * Esta classe é a interface gráfica da ferramenta. Foi construída com o apoio
 * do Netbeans Mobility Pack, portanto boa parte de seu código foi gerado por
 * tal ferramenta.
 *
 */
public class GerenciadorDeCertificadosMidlet extends MIDlet implements
```

```
CommandListener, Observador {

/** Creates a new instance of GerenciadorDeCertificadosMidlet */
public GerenciadorDeCertificadosMidlet() {
initialize();
Controlador.getInstance().registrarObservador(this);
}

private String reqAtual;
private int certAtual;
private List listPrincipal;
private Command sairDaAplicacaoCommand;
private Form formRequisicao;
private TextField textFieldCN;
private Command gerarRequisicaoVoltarCommand;
private Command gerarRequisicaoOkCommand;
private org.netbeans.microedition.lcdui.WaitScreen waitScreenGerandoRequisicao;
private Alert alertSucessoRequisicao;
private Alert alertErroRequisicao;
private List listCertificadosInstalados;
private Command listarCertificadosInstaladosVoltarCommand;
private Command informacoesDoCertificadoVoltarCommand;
private Command listarCertificadosInstaladosOkCommand;
private Form formConfiguracoes;
private TextField textFieldURLRequisicao;
private TextField textFieldURLInstalacaoDeCertificados;
private Command configuracoesOkCommand;
private Command configuracoesVoltarCommand;
private Command instalacaoDeCertificadoOkCommand;
private Command instalacaoDeCertificadoVoltarCommand;
private org.netbeans.microedition.lcdui.WaitScreen waitScreenInstalandoCertificado;
private Alert alertSucessoInstalacao;
private Alert alertErroInstalacao;
private org.netbeans.microedition.util.SimpleCancellableTask simpleCancellableTask1
private Command excluirCommand;
```

```
private org.netbeans.microedition.util.SimpleCancellableTask simpleCancellableTask2
private Command enviarRequisicaoCommand;
private Command okCommand1;
private org.netbeans.microedition.lcdui.WaitScreen waitScreenEnviandoRequisicao;
private Alert alertSucessoEnvioDaRequisicao;
private Command backCommand1;
private Command backCommand2;
private Command backCommand3;
private Form formEnvioDeRequisicao;
private TextField textFieldLogin;
private TextField textFieldSenha;
private Command okCommand2;
private Form formInformacoesDoCertificado;
private Command backCommand4;
private Command backCommand5;
private StringItem stringItemDN;
private StringItem stringItemSHA1;
private DateField dateFieldEmitidoEm;
private DateField dateFieldValidoAte;
private StringItem stringItemEmissorDN;
private TextField textFieldTamanhoDaChave;
private Command backCommand6;
private Command backCommand7;
private Alert alertSucessoExclusao;
private Command okCommand3;
private Alert alertConfirmacaoExclusao;
private Command okCommand4;
private Command backCommand8;
private List listRequisicoesPendentes;
private Command okCommand5;
private Command okCommand6;
private Command backCommand9;
private org.netbeans.microedition.util.SimpleCancellableTask simpleCancellableTask3

/**
```

```
 * Called by the system to indicate that a command has been invoked on a
 * particular displayable.
 *
 * @param command
 *            the Command that ws invoked
 * @param displayable
 *            the Displayable on which the command was invoked
 */
public void commandAction(Command command, Displayable displayable) {
// Insert global pre-action code here
if (displayable == listPrincipal) {
if (command == listPrincipal.SELECT_COMMAND) {
switch (get_listPrincipal().getSelectedIndex()) {
case 0:
// Insert pre-action code here
getDisplay().setCurrent(get_formRequisicao());
// Insert post-action code here
break;
case 1:
// Insert pre-action code here
getDisplay().setCurrent(get_listRequisicoesPendentes());
// Insert post-action code here
break;
case 2:
// Insert pre-action code here
listCertificadosInstalados = null;
getDisplay().setCurrent(get_listCertificadosInstalados());
// Insert post-action code here
break;
case 3:
// Insert pre-action code here

get_textFieldURLInstalacaoDeCertificados().setString(
Configuracao.getInstance()
.getUrlInstalacaoDeCertificado());
```

```
get_textFieldURLRequisicao().setString(
Configuracao.getInstance()
.getUrlEnvioDeRequisicao());
get_textFieldTamanhoDaChave().setString(
Integer.toString(Configuracao.getInstance()
.getTamanhoDaChave()));
getDisplay().setCurrent(get_formConfiguracoes());
// Insert post-action code here
Configuracao.getInstance().setUrlEnvioDeRequisicao(
get_textFieldURLRequisicao().getString());
Configuracao.getInstance().setUrlInstalacaoDeCertificado(
get_textFieldURLInstalacaoDeCertificados()
.getString());
Configuracao.getInstance().setTamanhoDaChave(
Integer.parseInt(get_textFieldSenha().getString()));
break;
}
} else if (command == sairDaAplicacaoCommand) {
// Insert pre-action code here
exitMIDlet();
// Insert post-action code here
}
} else if (displayable == formRequisicao) {
if (command == gerarRequisicaoOkCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_waitScreenGerandoRequisicao());
// Insert post-action code here
} else if (command == gerarRequisicaoVoltarCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-action code here
}
} else if (displayable == waitScreenGerandoRequisicao) {
if (command == waitScreenGerandoRequisicao.FAILURE_COMMAND) {
// Insert pre-action code here
```

```
getDisplay().setCurrent(get_alertErroRequisicao(),
get_listPrincipal());
// Insert post-action code here
} else if (command == waitScreenGerandoRequisicao.SUCCESS_COMMAND) {
// Insert pre-action code here
getDisplay().setCurrent(get_alertSucessoRequisicao(),
get_listPrincipal());
// Insert post-action code here
}
} else if (displayable == listCertificadosInstalados) {
if (command == listarCertificadosInstaladosVoltarCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-action code here
} else if (command == listarCertificadosInstaladosOkCommand) {
// Insert pre-action code here
certAtual = get_listCertificadosInstalados().getSelectedIndex();
getDisplay().setCurrent(get_formInformacoesDoCertificado());
// Insert post-action code here
Certificado c = (Certificado) Controlador.getInstance()
.obterCertificadosInstalados().elementAt(certAtual);
get_stringItemDN().setText(c.getDN());
get_stringItemSHA1().setText(c.getSHA1());
get_dateFieldEmitidoEm().setDate(c.getEmitidoEm());
get_dateFieldValidoAte().setDate(c.getValidoAte());
get_stringItemEmissorDN().setText(c.getEmissorDN());
}
} else if (displayable == formConfiguracoes) {
if (command == configuracoesOkCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-action code here
} else if (command == configuracoesVoltarCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
```

```
// Insert post-action code here
}
} else if (displayable == waitScreenInstalandoCertificado) {
if (command == waitScreenInstalandoCertificado.SUCCESS_COMMAND) {
// Insert pre-action code here
getDisplay().setCurrent(get_alertSucessoInstalacao(),
get_listPrincipal());
// Insert post-action code here
} else if (command == waitScreenInstalandoCertificado.FAILURE_COMMAND) {
// Insert pre-action code here
getDisplay().setCurrent(get_alertErroInstalacao(),
get_listPrincipal());
// Insert post-action code here
}
} else if (displayable == alertSucessoRequisicao) {
if (command == enviarRequisicaoCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_formEnvioDeRequisicao());
// Insert post-action code here
} else if (command == backCommand3) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-action code here
}
} else if (displayable == waitScreenEnviandoRequisicao) {
if (command == waitScreenEnviandoRequisicao.FAILURE_COMMAND) {
// Insert pre-action code here
get_alertErroRequisicao().setString(
"Erro ao enviar a requisi\u00E7\u00E3o.");
getDisplay().setCurrent(get_alertErroRequisicao(),
get_listPrincipal());
// Insert post-action code here
} else if (command == waitScreenEnviandoRequisicao.SUCCESS_COMMAND) {
// Insert pre-action code here
getDisplay().setCurrent(get_alertSucessoEnvioDaRequisicao(),
```

```
get_listPrincipal());
// Insert post-action code here
}
} else if (displayable == formEnvioDeRequisicao) {
if (command == okCommand2) {
// Insert pre-action code here
getDisplay().setCurrent(get_waitScreenEnviandoRequisicao());
// Insert post-action code here
}
} else if (displayable == formInformacoesDoCertificado) {
if (command == excluirCommand) {
// Insert pre-action code here
getDisplay().setCurrent(get_alertConfirmacaoExclusao(),
get_formInformacoesDoCertificado());
Certificado c = (Certificado) Controlador.getInstance()
.obterCertificadosInstalados().elementAt(certAtual);
get_alertConfirmacaoExclusao()
.setString(
c.toString()
+ " ser\u00E1 exclu\u00EDdo! Confirma a exclus\u00E3o?");
} else if (command == backCommand5) {
// Insert pre-action code here
getDisplay().setCurrent(get_listCertificadosInstalados());
// Insert post-action code here
}
} else if (displayable == alertConfirmacaoExclusao) {
if (command == excluirCommand) {
// Insert pre-action code here
Certificado c = (Certificado) Controlador.getInstance()
.obterCertificadosInstalados().elementAt(certAtual);
Controlador.getInstance().removerCertificado(c);
listCertificadosInstalados = null;
getDisplay().setCurrent(get_alertSucessoExclusao(),
get_listCertificadosInstalados());
// Insert post-action code here
```

```
} else if (command == backCommand8) {
// Insert pre-action code here
getDisplay().setCurrent(get_formInformacoesDoCertificado());
// Insert post-action code here
}
} else if (displayable == listRequisicoesPendentes) {
if (command == backCommand9) {
// Insert pre-action code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-action code here
} else if (command == okCommand6) {
// Insert pre-action code here
certAtual = get_listRequisicoesPendentes().getSelectedIndex();
getDisplay().setCurrent(get_waitScreenInstalandoCertificado());
// Insert post-action code here
}
}
// Insert global post-action code here
}

/**
 * This method initializes UI of the application.
 */
private void initialize() {
// Insert pre-init code here
getDisplay().setCurrent(get_listPrincipal());
// Insert post-init code here
}

/**
 * This method should return an instance of the display.
 */
public Display getDisplay() {
return Display.getDisplay(this);
}
```

```
/**
 * This method should exit the midlet.
 */
public void exitMIDlet() {
getDisplay().setCurrent(null);
destroyApp(true);
notifyDestroyed();
}

/**
 * This method returns instance for listPrincipal component and should be
 * called instead of accessing listPrincipal field directly.
 *
 * @return Instance for listPrincipal component
 */
public List get_listPrincipal() {
if (listPrincipal == null) {
// Insert pre-init code here
listPrincipal = new List(null, Choice.IMPLICIT,
new String[] { "Gerar Requisi\u00E7\u00E3o",
"Instalar Certificado",
"Listar Certificados Instalados",
"Configura\u00E7\u00F5es" }, new Image[] { null,
null, null, null });
listPrincipal.addCommand(get_sairDaAplicacaoCommand());
listPrincipal.setCommandListener(this);
listPrincipal.setSelectedFlags(new boolean[] { false, false, false,
false });
// Insert post-init code here
}
return listPrincipal;
}

/**
```

```
 * This method returns instance for sairDaAplicacaoCommand component and
 * should be called instead of accessing sairDaAplicacaoCommand field
 * directly.
 *
 * @return Instance for sairDaAplicacaoCommand component
 */
public Command get_sairDaAplicacaoCommand() {
if (sairDaAplicacaoCommand == null) {
// Insert pre-init code here
sairDaAplicacaoCommand = new Command("Exit", Command.EXIT, 1);
// Insert post-init code here
}
return sairDaAplicacaoCommand;
}

/**
 * This method returns instance for formRequisicao component and should be
 * called instead of accessing formRequisicao field directly.
 *
 * @return Instance for formRequisicao component
 */
public Form get_formRequisicao() {
if (formRequisicao == null) {
// Insert pre-init code here
formRequisicao = new Form(null, new Item[] { get_textFieldCN() });
formRequisicao.addCommand(get_gerarRequisicaoOkCommand());
formRequisicao.addCommand(get_gerarRequisicaoVoltarCommand());
formRequisicao.setCommandListener(this);
// Insert post-init code here
}
return formRequisicao;
}

/**
 * This method returns instance for textFieldCN component and should be
```

```
 * called instead of accessing textFieldCN field directly.
 *
 * @return Instance for textFieldCN component
 */
public TextField get_textFieldCN() {
if (textFieldCN == null) {
// Insert pre-init code here
textFieldCN = new TextField("Nome", null, 120, TextField.ANY);
// Insert post-init code here
}
return textFieldCN;
}

/**
 * This method returns instance for gerarRequisicaoVoltarCommand component
 * and should be called instead of accessing gerarRequisicaoVoltarCommand
 * field directly.
 *
 * @return Instance for gerarRequisicaoVoltarCommand component
 */
public Command get_gerarRequisicaoVoltarCommand() {
if (gerarRequisicaoVoltarCommand == null) {
// Insert pre-init code here
gerarRequisicaoVoltarCommand = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return gerarRequisicaoVoltarCommand;
}

/**
 * This method returns instance for gerarRequisicaoOkCommand component and
 * should be called instead of accessing gerarRequisicaoOkCommand field
 * directly.
 *
 * @return Instance for gerarRequisicaoOkCommand component
```

```
 */
public Command get_gerarRequisicaoOkCommand() {
if (gerarRequisicaoOkCommand == null) {
// Insert pre-init code here
gerarRequisicaoOkCommand = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return gerarRequisicaoOkCommand;
}

/**
 * This method returns instance for waitScreenGerandoRequisicao component
 * and should be called instead of accessing waitScreenGerandoRequisicao
 * field directly.
 *
 * @return Instance for waitScreenGerandoRequisicao component
 */
public org.netbeans.microedition.lcdui.WaitScreen get_waitScreenGerandoRequisicao()
if (waitScreenGerandoRequisicao == null) {
// Insert pre-init code here
waitScreenGerandoRequisicao = new org.netbeans.microedition.lcdui.WaitScreen(
getDisplay());
waitScreenGerandoRequisicao.setCommandListener(this);
waitScreenGerandoRequisicao
.setText("Gerando requisi\u00E7\u00E3o...");
waitScreenGerandoRequisicao.setTask(get_simpleCancellableTask1());
// Insert post-init code here
}
return waitScreenGerandoRequisicao;
}

/**
 * This method returns instance for alertSucessoRequisicao component and
 * should be called instead of accessing alertSucessoRequisicao field
 * directly.
```

```
 *
 * @return Instance for alertSucessoRequisicao component
 */
public Alert get_alertSucessoRequisicao() {
if (alertSucessoRequisicao == null) {
// Insert pre-init code here
alertSucessoRequisicao = new Alert(
null,
"Requisi\u00E7\u00E3o gerada com sucesso! Deseja envi\u00E1-la para a AC?",
null, null);
alertSucessoRequisicao.addCommand(get_enviarRequisicaoCommand());
alertSucessoRequisicao.addCommand(get_backCommand3());
alertSucessoRequisicao.setCommandListener(this);
alertSucessoRequisicao.setTimeout(-2);
// Insert post-init code here
}
return alertSucessoRequisicao;
}

/**
 * This method returns instance for alertErroRequisicao component and should
 * be called instead of accessing alertErroRequisicao field directly.
 *
 * @return Instance for alertErroRequisicao component
 */
public Alert get_alertErroRequisicao() {
if (alertErroRequisicao == null) {
// Insert pre-init code here
alertErroRequisicao = new Alert(null,
"Erro ao gerar a requisi\u00E7\u00E3o!", null, null);
alertErroRequisicao.setTimeout(-2);
// Insert post-init code here
}
return alertErroRequisicao;
}
```

```
/**
 * This method returns instance for listCertificadosInstalados component and
 * should be called instead of accessing listCertificadosInstalados field
 * directly.
 *
 * @return Instance for listCertificadosInstalados component
 */
public List get_listCertificadosInstalados() {
if (listCertificadosInstalados == null) {
// Insert pre-init code here
Vector vector = Controlador.getInstance()
.obterCertificadosInstalados();
listCertificadosInstalados = new List(null, Choice.IMPLICIT,
new String[0], new Image[0]);
listCertificadosInstalados
.addCommand(get_listarCertificadosInstaladosOkCommand());
listCertificadosInstalados
.addCommand(get_listarCertificadosInstaladosVoltarCommand());
listCertificadosInstalados.setCommandListener(this);
listCertificadosInstalados.setSelectedFlags(new boolean[0]);
// Insert post-init code here
for (int i = 0; i < vector.size(); i++) {
listCertificadosInstalados.append(vector.elementAt(i)
.toString(), null);
}
}
return listCertificadosInstalados;
}

/**
 * This method returns instance for
 * listarCertificadosInstaladosVoltarCommand component and should be called
 * instead of accessing listarCertificadosInstaladosVoltarCommand field
 * directly.
```

```
 *
 * @return Instance for listarCertificadosInstaladosVoltarCommand component
 */
public Command get_listarCertificadosInstaladosVoltarCommand() {
if (listarCertificadosInstaladosVoltarCommand == null) {
// Insert pre-init code here
listarCertificadosInstaladosVoltarCommand = new Command("Back",
Command.BACK, 1);
// Insert post-init code here
}
return listarCertificadosInstaladosVoltarCommand;
}

/**
 * This method returns instance for informacoesDoCertificadoVoltarCommand
 * component and should be called instead of accessing
 * informacoesDoCertificadoVoltarCommand field directly.
 *
 * @return Instance for informacoesDoCertificadoVoltarCommand component
 */
public Command get_informacoesDoCertificadoVoltarCommand() {
if (informacoesDoCertificadoVoltarCommand == null) {
// Insert pre-init code here
informacoesDoCertificadoVoltarCommand = new Command("Back",
Command.BACK, 1);
// Insert post-init code here
}
return informacoesDoCertificadoVoltarCommand;
}

/**
 * This method returns instance for listarCertificadosInstaladosOkCommand
 * component and should be called instead of accessing
 * listarCertificadosInstaladosOkCommand field directly.
 *
```

```
 * @return Instance for listarCertificadosInstaladosOkCommand component
 */
public Command get_listarCertificadosInstaladosOkCommand() {
if (listarCertificadosInstaladosOkCommand == null) {
// Insert pre-init code here
listarCertificadosInstaladosOkCommand = new Command("Ok",
Command.OK, 1);
// Insert post-init code here
}
return listarCertificadosInstaladosOkCommand;
}

/**
 * This method returns instance for formConfiguracoes component and should
 * be called instead of accessing formConfiguracoes field directly.
 *
 * @return Instance for formConfiguracoes component
 */
public Form get_formConfiguracoes() {
if (formConfiguracoes == null) {
// Insert pre-init code here
formConfiguracoes = new Form(null, new Item[] {
get_textFieldURLRequisicao(),
get_textFieldURLInstalacaoDeCertificados(),
get_textFieldTamanhoDaChave() });
formConfiguracoes.addCommand(get_configuracoesOkCommand());
formConfiguracoes.addCommand(get_configuracoesVoltarCommand());
formConfiguracoes.setCommandListener(this);
// Insert post-init code here
}
return formConfiguracoes;
}

/**
 * This method returns instance for textFieldURLRequisicao component and
```

```
 * should be called instead of accessing textFieldURLRequisicao field
 * directly.
 *
 * @return Instance for textFieldURLRequisicao component
 */
public TextField get_textFieldURLRequisicao() {
if (textFieldURLRequisicao == null) {
// Insert pre-init code here
textFieldURLRequisicao = new TextField(
"URL de Envio de Requisi\u00E7\u00E3o", null, 120,
TextField.URL);
// Insert post-init code here
textFieldURLRequisicao.setString(Configuracao.getInstance()
.getUrlEnvioDeRequisicao());
}
return textFieldURLRequisicao;
}

/**
 * This method returns instance for textFieldURLInstalacaoDeCertificados
 * component and should be called instead of accessing
 * textFieldURLInstalacaoDeCertificados field directly.
 *
 * @return Instance for textFieldURLInstalacaoDeCertificados component
 */
public TextField get_textFieldURLInstalacaoDeCertificados() {
if (textFieldURLInstalacaoDeCertificados == null) {
// Insert pre-init code here
textFieldURLInstalacaoDeCertificados = new TextField(
"URL de Instala\u00E7\u00E3o de Certificados", null, 120,
TextField.URL);
// Insert post-init code here
textFieldURLInstalacaoDeCertificados.setString(Configuracao
.getInstance().getUrlInstalacaoDeCertificado());
}
```

```
return textFieldURLInstalacaoDeCertificados;
}

/**
 * This method returns instance for configuracoesOkCommand component and
 * should be called instead of accessing configuracoesOkCommand field
 * directly.
 *
 * @return Instance for configuracoesOkCommand component
 */
public Command get_configuracoesOkCommand() {
if (configuracoesOkCommand == null) {
// Insert pre-init code here
configuracoesOkCommand = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return configuracoesOkCommand;
}

/**
 * This method returns instance for configuracoesVoltarCommand component and
 * should be called instead of accessing configuracoesVoltarCommand field
 * directly.
 *
 * @return Instance for configuracoesVoltarCommand component
 */
public Command get_configuracoesVoltarCommand() {
if (configuracoesVoltarCommand == null) {
// Insert pre-init code here
configuracoesVoltarCommand = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return configuracoesVoltarCommand;
}
```

```
/**
 * This method returns instance for instalacaoDeCertificadoOkCommand
 * component and should be called instead of accessing
 * instalacaoDeCertificadoOkCommand field directly.
 *
 * @return Instance for instalacaoDeCertificadoOkCommand component
 */
public Command get_instalacaoDeCertificadoOkCommand() {
if (instalacaoDeCertificadoOkCommand == null) {
// Insert pre-init code here
instalacaoDeCertificadoOkCommand = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return instalacaoDeCertificadoOkCommand;
}

/**
 * This method returns instance for instalacaoDeCertificadoVoltarCommand
 * component and should be called instead of accessing
 * instalacaoDeCertificadoVoltarCommand field directly.
 *
 * @return Instance for instalacaoDeCertificadoVoltarCommand component
 */
public Command get_instalacaoDeCertificadoVoltarCommand() {
if (instalacaoDeCertificadoVoltarCommand == null) {
// Insert pre-init code here
instalacaoDeCertificadoVoltarCommand = new Command("Back",
Command.BACK, 1);
// Insert post-init code here
}
return instalacaoDeCertificadoVoltarCommand;
}

/**
 * This method returns instance for waitScreenInstalandoCertificado
```

```
 * component and should be called instead of accessing
 * waitScreenInstalandoCertificado field directly.
 *
 * @return Instance for waitScreenInstalandoCertificado component
 */
public org.netbeans.microedition.lcdui.WaitScreen get_waitScreenInstalandoCertifica
if (waitScreenInstalandoCertificado == null) {
// Insert pre-init code here
waitScreenInstalandoCertificado = new org.netbeans.microedition.lcdui.WaitScreen(
getDisplay());
waitScreenInstalandoCertificado.setCommandListener(this);
waitScreenInstalandoCertificado.setText("Intalando certificado...");
waitScreenInstalandoCertificado
.setTask(get_simpleCancellableTask3());
// Insert post-init code here
}
return waitScreenInstalandoCertificado;
}

/**
 * This method returns instance for alertSucessoInstalacao component and
 * should be called instead of accessing alertSucessoInstalacao field
 * directly.
 *
 * @return Instance for alertSucessoInstalacao component
 */
public Alert get_alertSucessoInstalacao() {
if (alertSucessoInstalacao == null) {
// Insert pre-init code here
alertSucessoInstalacao = new Alert(null,
"Certificado instalado com sucesso!", null, null);
alertSucessoInstalacao.setTimeout(-2);
// Insert post-init code here
}
return alertSucessoInstalacao;
```

```
}

/**
 * This method returns instance for alertErroInstalacao component and should
 * be called instead of accessing alertErroInstalacao field directly.
 *
 * @return Instance for alertErroInstalacao component
 */
public Alert get_alertErroInstalacao() {
if (alertErroInstalacao == null) {
// Insert pre-init code here
alertErroInstalacao = new Alert(null,
"Erro na instala\u00E7\u00E3o do certificado!", null, null);
alertErroInstalacao.setTimeout(-2);
// Insert post-init code here
}
return alertErroInstalacao;
}

/**
 * This method returns instance for simpleCancellableTask1 component and
 * should be called instead of accessing simpleCancellableTask1 field
 * directly.
 *
 * @return Instance for simpleCancellableTask1 component
 */
public org.netbeans.microedition.util.SimpleCancellableTask get_simpleCancellableTa
if (simpleCancellableTask1 == null) {
// Insert pre-init code here
simpleCancellableTask1 = new org.netbeans.microedition.util.SimpleCancellableTask()
simpleCancellableTask1
.setExecutable(new org.netbeans.microedition.util.Executable() {
public void execute() throws Exception {
String cn = textFieldCN.getString();
reqAtual = Controlador.getInstance()
```

```
.gerarRequisicao(cn);
}
});
// Insert post-init code here
}
return simpleCancellableTask1;
}

/**
 * This method returns instance for excluirCommand component and should be
 * called instead of accessing excluirCommand field directly.
 *
 * @return Instance for excluirCommand component
 */
public Command get_excluirCommand() {
if (excluirCommand == null) {
// Insert pre-init code here
excluirCommand = new Command("Excluir", Command.OK, 1);
// Insert post-init code here
}
return excluirCommand;
}

/**
 * This method returns instance for simpleCancellableTask2 component and
 * should be called instead of accessing simpleCancellableTask2 field
 * directly.
 *
 * @return Instance for simpleCancellableTask2 component
 */
public org.netbeans.microedition.util.SimpleCancellableTask get_simpleCancellableTa
if (simpleCancellableTask2 == null) {
// Insert pre-init code here
simpleCancellableTask2 = new org.netbeans.microedition.util.SimpleCancellableTask()
simpleCancellableTask2
```

```
.setExecutable(new org.netbeans.microedition.util.Executable() {
public void execute() throws Exception {
Controlador.getInstance().enviarRequisicao(
get_textFieldLogin().getString(),
get_textFieldSenha().getString(), reqAtual);
}
});
// Insert post-init code here
}
return simpleCancellableTask2;
}

/**
 * This method returns instance for enviarRequisicaoCommand component and
 * should be called instead of accessing enviarRequisicaoCommand field
 * directly.
 *
 * @return Instance for enviarRequisicaoCommand component
 */
public Command get_enviarRequisicaoCommand() {
if (enviarRequisicaoCommand == null) {
// Insert pre-init code here
enviarRequisicaoCommand = new Command("Enviar", Command.OK, 1);
// Insert post-init code here
}
return enviarRequisicaoCommand;
}

/**
 * This method returns instance for okCommand1 component and should be
 * called instead of accessing okCommand1 field directly.
 *
 * @return Instance for okCommand1 component
 */
public Command get_okCommand1() {
```

```
if (okCommand1 == null) {
// Insert pre-init code here
okCommand1 = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return okCommand1;
}

/**
 * This method returns instance for waitScreenEnviandoRequisicao component
 * and should be called instead of accessing waitScreenEnviandoRequisicao
 * field directly.
 *
 * @return Instance for waitScreenEnviandoRequisicao component
 */
public org.netbeans.microedition.lcdui.WaitScreen get_waitScreenEnviandoRequisicao(
if (waitScreenEnviandoRequisicao == null) {
// Insert pre-init code here
waitScreenEnviandoRequisicao = new org.netbeans.microedition.lcdui.WaitScreen(
getDisplay());
waitScreenEnviandoRequisicao.setCommandListener(this);
waitScreenEnviandoRequisicao
.setText("Enviando a requisi\u00E7\u00E3o...");
waitScreenEnviandoRequisicao.setTask(get_simpleCancellableTask2());
// Insert post-init code here
}
return waitScreenEnviandoRequisicao;
}

/**
 * This method returns instance for alertSucessoEnvioDaRequisicao component
 * and should be called instead of accessing alertSucessoEnvioDaRequisicao
 * field directly.
 *
 * @return Instance for alertSucessoEnvioDaRequisicao component
```

```
 */
public Alert get_alertSucessoEnvioDaRequisicao() {
if (alertSucessoEnvioDaRequisicao == null) {
// Insert pre-init code here
alertSucessoEnvioDaRequisicao = new Alert(null,
"Certificado obtido com sucesso!", null, null);
alertSucessoEnvioDaRequisicao.setTimeout(-2);
// Insert post-init code here
}
return alertSucessoEnvioDaRequisicao;
}

/**
 * This method returns instance for backCommand1 component and should be
 * called instead of accessing backCommand1 field directly.
 *
 * @return Instance for backCommand1 component
 */
public Command get_backCommand1() {
if (backCommand1 == null) {
// Insert pre-init code here
backCommand1 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand1;
}

/**
 * This method returns instance for backCommand2 component and should be
 * called instead of accessing backCommand2 field directly.
 *
 * @return Instance for backCommand2 component
 */
public Command get_backCommand2() {
if (backCommand2 == null) {
```

```
// Insert pre-init code here
backCommand2 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand2;
}

/**
 * This method returns instance for backCommand3 component and should be
 * called instead of accessing backCommand3 field directly.
 *
 * @return Instance for backCommand3 component
 */
public Command get_backCommand3() {
if (backCommand3 == null) {
// Insert pre-init code here
backCommand3 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand3;
}

/**
 * This method returns instance for formEnvioDeRequisicao component and
 * should be called instead of accessing formEnvioDeRequisicao field
 * directly.
 *
 * @return Instance for formEnvioDeRequisicao component
 */
public Form get_formEnvioDeRequisicao() {
if (formEnvioDeRequisicao == null) {
// Insert pre-init code here
formEnvioDeRequisicao = new Form(null, new Item[] {
get_textFieldLogin(), get_textFieldSenha() });
formEnvioDeRequisicao.addCommand(get_okCommand2());
```

```
formEnvioDeRequisicao.setCommandListener(this);
// Insert post-init code here
}
return formEnvioDeRequisicao;
}

/**
 * This method returns instance for textFieldLogin component and should be
 * called instead of accessing textFieldLogin field directly.
 *
 * @return Instance for textFieldLogin component
 */
public TextField get_textFieldLogin() {
if (textFieldLogin == null) {
// Insert pre-init code here
textFieldLogin = new TextField("Login", null, 120, TextField.ANY);
// Insert post-init code here
}
return textFieldLogin;
}

/**
 * This method returns instance for textFieldSenha component and should be
 * called instead of accessing textFieldSenha field directly.
 *
 * @return Instance for textFieldSenha component
 */
public TextField get_textFieldSenha() {
if (textFieldSenha == null) {
// Insert pre-init code here
textFieldSenha = new TextField("Senha", null, 120, TextField.ANY
| TextField.PASSWORD);
// Insert post-init code here
}
return textFieldSenha;
```

```
}

/**
 * This method returns instance for okCommand2 component and should be
 * called instead of accessing okCommand2 field directly.
 *
 * @return Instance for okCommand2 component
 */
public Command get_okCommand2() {
if (okCommand2 == null) {
// Insert pre-init code here
okCommand2 = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return okCommand2;
}

/**
 * This method returns instance for formInformacoesDoCertificado component
 * and should be called instead of accessing formInformacoesDoCertificado
 * field directly.
 *
 * @return Instance for formInformacoesDoCertificado component
 */
public Form get_formInformacoesDoCertificado() {
if (formInformacoesDoCertificado == null) {
// Insert pre-init code here
formInformacoesDoCertificado = new Form(null, new Item[] {
get_stringItemDN(), get_stringItemEmissorDN(),
get_dateFieldEmitidoEm(), get_dateFieldValidoAte(),
get_stringItemSHA1() });
formInformacoesDoCertificado.addCommand(get_excluirCommand());
formInformacoesDoCertificado.addCommand(get_backCommand5());
formInformacoesDoCertificado.setCommandListener(this);
// Insert post-init code here
```

```
}
return formInformacoesDoCertificado;
}

/**
 * This method returns instance for backCommand4 component and should be
 * called instead of accessing backCommand4 field directly.
 *
 * @return Instance for backCommand4 component
 */
public Command get_backCommand4() {
if (backCommand4 == null) {
// Insert pre-init code here
backCommand4 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand4;
}

/**
 * This method returns instance for backCommand5 component and should be
 * called instead of accessing backCommand5 field directly.
 *
 * @return Instance for backCommand5 component
 */
public Command get_backCommand5() {
if (backCommand5 == null) {
// Insert pre-init code here
backCommand5 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand5;
}

/**
```

```
 * This method returns instance for stringItemDN component and should be
 * called instead of accessing stringItemDN field directly.
 *
 * @return Instance for stringItemDN component
 */
public StringItem get_stringItemDN() {
if (stringItemDN == null) {
// Insert pre-init code here
stringItemDN = new StringItem("Emitido para:", "<Enter Text>");
// Insert post-init code here
}
return stringItemDN;
}

/**
 * This method returns instance for stringItemSHA1 component and should be
 * called instead of accessing stringItemSHA1 field directly.
 *
 * @return Instance for stringItemSHA1 component
 */
public StringItem get_stringItemSHA1() {
if (stringItemSHA1 == null) {
// Insert pre-init code here
stringItemSHA1 = new StringItem("SHA1:", "<Enter Text>");
// Insert post-init code here
}
return stringItemSHA1;
}

/**
 * This method returns instance for dateFieldEmitidoEm component and should
 * be called instead of accessing dateFieldEmitidoEm field directly.
 *
 * @return Instance for dateFieldEmitidoEm component
 */
```

```
public DateField get_dateFieldEmitidoEm() {
if (dateFieldEmitidoEm == null) {
// Insert pre-init code here
dateFieldEmitidoEm = new DateField("Emitido em:", DateField.DATE);
// Insert post-init code here
}
return dateFieldEmitidoEm;
}

/**
 * This method returns instance for dateFieldValidoAte component and should
 * be called instead of accessing dateFieldValidoAte field directly.
 *
 * @return Instance for dateFieldValidoAte component
 */
public DateField get_dateFieldValidoAte() {
if (dateFieldValidoAte == null) {
// Insert pre-init code here
dateFieldValidoAte = new DateField("V\u00E1lido at\u00E9:",
DateField.DATE);
// Insert post-init code here
}
return dateFieldValidoAte;
}

/**
 * This method returns instance for stringItemEmissorDN component and should
 * be called instead of accessing stringItemEmissorDN field directly.
 *
 * @return Instance for stringItemEmissorDN component
 */
public StringItem get_stringItemEmissorDN() {
if (stringItemEmissorDN == null) {
// Insert pre-init code here
stringItemEmissorDN = new StringItem("Emitido por:", "<Enter Text>");
```

```
// Insert post-init code here
}
return stringItemEmissorDN;
}

/**
 * This method returns instance for textFieldTamanhoDaChave component and
 * should be called instead of accessing textFieldTamanhoDaChave field
 * directly.
 *
 * @return Instance for textFieldTamanhoDaChave component
 */
public TextField get_textFieldTamanhoDaChave() {
if (textFieldTamanhoDaChave == null) {
// Insert pre-init code here
textFieldTamanhoDaChave = new TextField("Tamanho da chave", null,
120, TextField.ANY);
// Insert post-init code here
}
return textFieldTamanhoDaChave;
}

/**
 * This method returns instance for backCommand6 component and should be
 * called instead of accessing backCommand6 field directly.
 *
 * @return Instance for backCommand6 component
 */
public Command get_backCommand6() {
if (backCommand6 == null) {
// Insert pre-init code here
backCommand6 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand6;
```

```
}

/**
 * This method returns instance for backCommand7 component and should be
 * called instead of accessing backCommand7 field directly.
 *
 * @return Instance for backCommand7 component
 */
public Command get_backCommand7() {
if (backCommand7 == null) {
// Insert pre-init code here
backCommand7 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand7;
}

/**
 * This method returns instance for alertSucessoExclusao component and
 * should be called instead of accessing alertSucessoExclusao field
 * directly.
 *
 * @return Instance for alertSucessoExclusao component
 */
public Alert get_alertSucessoExclusao() {
if (alertSucessoExclusao == null) {
// Insert pre-init code here
alertSucessoExclusao = new Alert(null,
"Exclu\u00EDdo com sucesso!", null, null);
alertSucessoExclusao.setTimeout(-2);
// Insert post-init code here
}
return alertSucessoExclusao;
}
```

```
/**
 * This method returns instance for okCommand3 component and should be
 * called instead of accessing okCommand3 field directly.
 *
 * @return Instance for okCommand3 component
 */
public Command get_okCommand3() {
if (okCommand3 == null) {
// Insert pre-init code here
okCommand3 = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return okCommand3;
}

/**
 * This method returns instance for alertConfirmacaoExclusao component and
 * should be called instead of accessing alertConfirmacaoExclusao field
 * directly.
 *
 * @return Instance for alertConfirmacaoExclusao component
 */
public Alert get_alertConfirmacaoExclusao() {
if (alertConfirmacaoExclusao == null) {
// Insert pre-init code here
alertConfirmacaoExclusao = new Alert(null,
"ser\u00E1 exclu\u00EDdo! Confirma a exclus\u00E3o?", null,
null);
alertConfirmacaoExclusao.addCommand(get_excluirCommand());
alertConfirmacaoExclusao.addCommand(get_backCommand8());
alertConfirmacaoExclusao.setCommandListener(this);
alertConfirmacaoExclusao.setTimeout(-2);
// Insert post-init code here
}
return alertConfirmacaoExclusao;
```

```
}

/**
 * This method returns instance for okCommand4 component and should be
 * called instead of accessing okCommand4 field directly.
 *
 * @return Instance for okCommand4 component
 */
public Command get_okCommand4() {
if (okCommand4 == null) {
// Insert pre-init code here
okCommand4 = new Command("Excluir", Command.OK, 1);
// Insert post-init code here
}
return okCommand4;
}

/**
 * This method returns instance for backCommand8 component and should be
 * called instead of accessing backCommand8 field directly.
 *
 * @return Instance for backCommand8 component
 */
public Command get_backCommand8() {
if (backCommand8 == null) {
// Insert pre-init code here
backCommand8 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand8;
}

/**
 * This method returns instance for listRequisicoesPendentes component and
 * should be called instead of accessing listRequisicoesPendentes field
```

```
 * directly.
 *
 * @return Instance for listRequisicoesPendentes component
 */
public List get_listRequisicoesPendentes() {
if (listRequisicoesPendentes == null) {
// Insert pre-init code here
Vector vector = Controlador.getInstance()
.obterRequisicoesPendentes();
listRequisicoesPendentes = new List(null, Choice.IMPLICIT,
new String[0], new Image[0]);
listRequisicoesPendentes.addCommand(get_okCommand6());
listRequisicoesPendentes.addCommand(get_backCommand9());
listRequisicoesPendentes.setCommandListener(this);
listRequisicoesPendentes.setSelectedFlags(new boolean[0]);
// Insert post-init code here
for (int i = 0; i < vector.size(); i++) {
listRequisicoesPendentes.append(vector.elementAt(i).toString(),
null);
}
}
return listRequisicoesPendentes;
}

/**
 * This method returns instance for okCommand5 component and should be
 * called instead of accessing okCommand5 field directly.
 *
 * @return Instance for okCommand5 component
 */
public Command get_okCommand5() {
if (okCommand5 == null) {
// Insert pre-init code here
okCommand5 = new Command("Ok", Command.OK, 1);
// Insert post-init code here
```

```
}
return okCommand5;
}

/**
 * This method returns instance for okCommand6 component and should be
 * called instead of accessing okCommand6 field directly.
 *
 * @return Instance for okCommand6 component
 */
public Command get_okCommand6() {
if (okCommand6 == null) {
// Insert pre-init code here
okCommand6 = new Command("Ok", Command.OK, 1);
// Insert post-init code here
}
return okCommand6;
}

/**
 * This method returns instance for backCommand9 component and should be
 * called instead of accessing backCommand9 field directly.
 *
 * @return Instance for backCommand9 component
 */
public Command get_backCommand9() {
if (backCommand9 == null) {
// Insert pre-init code here
backCommand9 = new Command("Back", Command.BACK, 1);
// Insert post-init code here
}
return backCommand9;
}

/**
```

```
 * This method returns instance for simpleCancellableTask3 component and
 * should be called instead of accessing simpleCancellableTask3 field
 * directly.
 *
 * @return Instance for simpleCancellableTask3 component
 */
public org.netbeans.microedition.util.SimpleCancellableTask get_simpleCancellableTa
if (simpleCancellableTask3 == null) {
// Insert pre-init code here
simpleCancellableTask3 = new org.netbeans.microedition.util.SimpleCancellableTask()
simpleCancellableTask3
.setExecutable(new org.netbeans.microedition.util.Executable() {
public void execute() throws Exception {
Certificado c = (Certificado) Controlador
.getInstance()
.obterCertificadosInstalados().elementAt(
certAtual);
Certificado certificado = Controlador.getInstance()
.baixarCertificado(c.getDN());
c.setEstado(Estado.CERTIFICADO_INSTALADO);
Controlador.getInstance().instalarCertificado(c);
}
});
// Insert post-init code here
}
return simpleCancellableTask3;
}

public void startApp() {
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
```

```
}

public void notificar(Evento evento) {
get_waitScreenEnviandoRequisicao().setText(evento.toString());
get_waitScreenGerandoRequisicao().setText(evento.toString());
get_waitScreenInstalandoCertificado().setText(evento.toString());
}
}
```