FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

ROBSON DE SOUSA MARTINS

16SCJ

INTEGRAÇÃO DE SISTEMAS:

SIMPLIFICANDO A VIDA DO CIDADÃO

ROBSON DE SOUSA MARTINS

RM 42055

São Paulo

2012

ROBSON DE SOUSA MARTINS

16SCJ

INTEGRAÇÃO DE SISTEMAS:

SIMPLIFICANDO A VIDA DO CIDADÃO

Trabalho de Conclusão de Curso apresentado à Faculdade de Informática e Administração Paulista, como um dos requisitos para conclusão do Curso de Pós-Graduação em Desenvolvimento de Aplicações Corporativas em Java/SOA

Orientador: Prof. Ms. Matheus Haddad

São Paulo

2012

Dedicatória: Este trabalho de conclusão de curso é dedicado a meus pais, a meu irmão e a minha querida esposa, pelo apoio emocional e compreensão nos momentos de dedicação à elaboração deste. Também o dedico ao meu lindo filho Felipe, que tem crescido e me surpreendido diariamente, muito mais do que a fascinação que a tecnologia sempre me proporcionou.

AGRADECIMENTOS

Agradeço a Deus, pois Ele é o Único digno de receber toda a glória, toda honra e todo o louvor. Sou grato aos professores da FIAP pelo empenho no ensino das disciplinas do curso, destacando aqui o orientador deste trabalho de conclusão, Matheus Haddad. Agradecimentos especiais a Adriana Del Nero, pelo incentivo e companheirismo, e uma fonte de inspiração para o tema deste trabalho.

RESUMO

Os sistemas de informação são ferramentas essenciais, tanto nas empresas privadas, bem como nos órgãos e entidades governamentais – pois permitem oferecer melhores serviços à sociedade, com custos mais baixos. Diante desse cenário, cada empresa ou órgão governamental possui um sistema próprio que atua no domínio de suas informações, de forma descentralizada. Para o cidadão, essa questão produz algumas dificuldades, como a necessidade de acesso a vários sistemas e autenticação através de diferentes cartões, números e senhas. Visando uma relação mais ágil e eficiente entre os órgãos de governo e o cidadão, e entre as empresas e seus clientes, este trabalho propõe a integração de diferentes sistemas de informação, com o uso de padrões já estabelecidos no Governo Brasileiro e no mercado de tecnologia, tais como: Arquitetura Orientada a Serviços (SOA), cartão eletrônico de identificação do cidadão, certificados digitais e outros - de forma que, com um único cartão, o cidadão se autentique de maneira segura e acesse diversos serviços oferecidos pelo governo e por empresas que aderirem ao modelo. Essa integração de sistemas poderá trazer muitos benefícios às entidades do governo, às empresas privadas e ao cidadão, oferecendo facilidade no acesso a serviços governamentais e privados, além de uma possível melhoria desses serviços.

Palavras-chave: Integração. Serviço. Governo. Cidadão. Certificado Digital. Cartão Único de Identificação.

LISTA DE ILUSTRAÇÕES

FIGURAS

QUADROS

SUMÁRIO

1 INTRODUÇÃO

Esta pesquisa se caracteriza como Trabalho de Conclusão de Curso, apresentado à Faculdade de Informática e Administração Paulista (FIAP) como um dos requisitos para conclusão do Curso de Pós-Graduação em Desenvolvimento de Aplicações Corporativas em Java/SOA (MBA/SCJ).

Ao longo deste trabalho, serão apresentadas as justificativas para a pesquisa e a delimitação de seu tema. Em seguida, será exposto o referencial teórico, e por fim, o desenvolvimento de uma prova de conceito (protótipo), como forma de comprovação prática do tema abordado.

Este trabalho está organizado da seguinte forma: o primeiro capítulo apresenta a delimitação do tema, as justificativas para pesquisa e seu objetivo; o segundo capítulo expõe o referencial teórico usado como base no desenvolvimento do trabalho. Já o terceiro capítulo, detalha a construção da prova de conceito (protótipo), através dos artefatos de engenharia de *software* (documentos de visão e arquitetura). Os principais trechos de código-fonte que compõem a implementação dessa prova de conceito são revelados no capítulo quatro. Informações sobre instalação e implantação do projeto estão presentes no capítulo cinco. O sexto capítulo traz a conclusão do trabalho e apresenta sugestões de futuras pesquisas que poderão derivar deste estudo. Alguns apêndices trazem informações adicionais sobre o desenvolvimento deste trabalho e completam a pesquisa.

1.1 Delimitação do Tema

A constante busca das empresas por competitividade, otimização de custos e satisfação dos clientes, tem impulsionado a implantação de sistemas informatizados dentro de seus departamentos. Os sistemas de informação têm se tornado ferramentas essenciais, tanto nos setores de gestão como nos setores produtivos.

Segundo Lealdine (2007):

> A informática está presente em todos os setores da economia e é capaz de definir o valor de pessoas, procedimentos, organizações, setores, empresas e países [...] Ao longo da revolução tecnológica, que teve início por volta dos anos 60, as empresas aos poucos estão se adaptando a esse meio, tendo que procurar sistemas mais avançados que lhes proporcionassem maior facilidade em administrar e organizar suas tarefas.

Além das empresas privadas, órgãos governamentais também têm se beneficiado da utilização de sistemas informatizados para otimizar seus processos, permitindo oferecer melhores serviços à sociedade, a custos mais baixos, conforme descrito no documento “Padrões de Interoperabilidade de Governo Eletrônico” (e-PING), escrito pelo Ministério do Planejamento, Orçamento e Gestão (BRASIL, 2012).

Diante desse cenário de informatização, cada empresa ou órgão governamental tem buscado implantar sistemas que atuem no domínio de suas informações, melhorando a governança sobre elas. Entretanto, de uma forma geral, cada uma dessas empresas ou entidades de governo possui um sistema próprio e descentralizado (GALLIERS, SUTHERLAND, 1991; GUIMARÃES, 2009).

Para o cidadão, essa questão produz algumas dificuldades: por exemplo, para usar um serviço bancário, ele precisa acessar um sistema, para verificar a regularidade de sua documentação, outro sistema. Para utilizar serviços públicos de saúde, mais um sistema. E em cada um desses sistemas, é necessário se autenticar de uma forma (com diferentes cartões, números e senhas).

O documento e-PING também menciona que “um governo moderno e integrado exige sistemas igualmente modernos e integrados, interoperáveis, trabalhando de forma íntegra, segura e coerente em todo o setor público” (BRASIL, 2012, p. 6).

Desta forma, a integração de sistemas se torna uma preocupação importante não só entre empresas e seus parceiros e clientes, mas também entre órgãos governamentais e a sociedade (BRASIL, 2003).

Uma possível abordagem para integração dos sistemas é a utilização de Arquitetura Orientada a Serviços (SOA), pois a interoperabilidade é uma de suas importantes características (ERL, 2005).

Segundo Draeger (2008) e Roch (2006), a utilização de SOA nas organizações pode trazer mais agilidade na resposta a mudanças (seja de mercado,

de legislação ou de missão), além de reduzir os custos necessários a essas mudanças.

A arquitetura SOA é focada em padrões abertos e amplamente reconhecidos pelo mercado, tais como *Simple Object Access Protocol* (SOAP) (W3C, 2007), *Web Services Description Language* (WSDL) (W3C, 2001), e utilizando tecnologias como *Hypertext Transfer Protocol* (HTTP), *eXtensible Markup Language* (XML) e outras para garantir a transferência de informações entre diferentes sistemas.

Além disso, a utilização de SOA é uma recomendação no e-PING, um documento elaborado por especialistas em Tecnologia da Informação e Comunicação, a partir de uma demanda do Ministério do Planejamento, Orçamento e Gestão do Governo (BRASIL, 2012).

As questões de segurança relacionadas a autenticação, podem ser solucionadas através do uso de identificação eletrônica (certificados digitais), conforme recomendação da "Infraestrutura de Chaves Públicas do Brasil" (ICP-Brasil), descrita no portal do Instituto Nacional de Tecnologia da Informação (BRASIL, 2011). Ainda segundo essa recomendação, o uso de certificados digitais armazenados em dispositivos criptográficos como *tokens* ou *smart cards*[1] está sendo gradativamente adotado em diversas entidades públicas, como Receita Federal do Brasil (RFB), Programa Universidade para Todos (ProUni), Agência Nacional de Saúde Suplementar (ANS), Instituto Nacional de Propriedade Industrial (INPI), entidades do Poder Judiciário, dentre outras.

Ainda, a emissão de documentos como o Cadastro de Pessoa Física - Eletrônico (e-CPF), o Cadastro Nacional de Pessoa Jurídica - Eletrônico (e-CNPJ), conforme publicado na instrução normativa da Secretaria da Receita Federal do Brasil (BRASIL, 1999), ou o Registro de Identidade Civil (RIC), como apresentado pelo Ministério da Justiça (BRASIL, 2010), já prevê a utilização de certificados digitais para identificar com segurança um cidadão, podendo essa característica ser aproveitada para a efetivação de autenticação integrada em vários sistemas de informação distintos.

---

[1] *Tokens* e *smart cards* são dispositivos portáteis que funcionam como mídias armazenadoras. Em seus *chips* são inseridas as chaves privadas dos usuários. O acesso às informações é feito por meio de uma senha pessoal. O *smart card* assemelha-se a um cartão magnético, sendo necessário um aparelho leitor para seu funcionamento. Já o *token* assemelha-se a uma pequena chave que é colocada em uma entrada do computador. Fonte: Instituto Nacional de Tecnologia da Informação (BRASIL, 2011).

1.2 Justificativas para a Pesquisa

A importância da integração de sistemas na esfera governamental, melhorando a relação e a transparência com o cidadão é indiscutível no ramo de atuação do pesquisador, visto que seus conhecimentos são aplicados em uma empresa cujo objetivo é desenvolver sistemas de informação para órgãos e entidades do governo. Além disso, no papel de cidadão, um de seus desejos é ter praticidade nas interações com os serviços púbicos e privados, especialmente reduzindo o número de cartões, números e senhas a decorar, sistemas a serem acessados, etc.

Essa integração de sistemas poderá trazer muitos benefícios às entidades do governo, como otimização de recursos, agilidade na resposta a mudanças de legislação, transparência governamental, dentre outros.

As empresas privadas também poderão se beneficiar de um cenário de integração, oferecendo maior qualidade aos seus clientes, diminuição de custos, redução de burocracia no relacionamento com entidades governamentais (como necessidades de recolhimento de impostos, emissão de documentação, etc.).

Já para o cidadão, a integração de sistemas e a arquitetura orientada a serviços serão traduzidas como facilidade no acesso a serviços governamentais e privados, além de uma possível redução de custos desses serviços. Além disso, como benefício indireto, a sua relação com o governo seria mais transparente, pois os recursos públicos poderiam ser melhor e mais claramente aplicados em benefício da sociedade, e o cidadão poderia ter acesso fácil a essas informações, conforme apresentado pelo artigo Democracia 3.0 (BRASIL, 2011).

1.3 Problema

Como realizar a integração entre diferentes sistemas governamentais e privados com o objetivo de simplificar a vida do cidadão?

1.4 Hipótese

Com base na questão (problema) central, a pesquisa apresenta a seguinte hipótese de trabalho: ao utilizar SOA e tratar das questões relativas à segurança da informação com o uso de certificados digitais, será possível integrar diferentes sistemas.

1.5 Objetivos

O objetivo deste trabalho é propor e implementar uma prova de conceito (protótipo), com a intenção de demonstrar uma possível integração entre sistemas, usando alguns padrões tecnológicos e governamentais já estabelecidos. A autenticação e autorização de um cidadão nesses sistemas realizar-se-á por meio de um cartão de identificação único.

2 REFERENCIAL TEÓRICO

2.1 Tecnologias de Informação e Comunicação

Algumas tecnologias - utilizadas no mercado e padronizadas por diferentes organizações e comitês - podem ser usadas para integrar sistemas e garantir a segurança das informações envolvidas nesse contexto de integração. Muitas delas são adotadas e recomendadas pelo documento e-PING (BRASIL, 2012).

A seguir, serão apresentadas algumas dessas tecnologias.

2.1.1 Arquitetura Orientada a Serviços

"A Arquitetura Orientada a Serviços (SOA) é um paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários" (OASIS, 2006).

Através de SOA, é possível organizar as atividades relevantes ao negócio de uma corporação (ou mesmo entre corporações parceiras), de modo a otimizar o fluxo, reuso e interoperabilidade entre essas atividades - sejam elas desempenhadas por sistemas computacionais ou por elementos humanos.

Com relação a sistemas de *software*, SOA provê - através do uso de alguns padrões tecnológicos - um mecanismo, chamado de serviço, para dar acesso a uma ou mais capacidades de um sistema computacional. O acesso a essas capacidades é realizado por meio de uma interface pré-definida e conhecida pelo utilizador do serviço, e sua execução deve seguir as restrições e políticas preestabelecidas.

Algumas características técnicas de uma implementação SOA são: baixo acoplamento entre serviços (menor impacto nas mudanças de regras de negócio), alta coesão (responsabilidades bem definidas), abstração (interfaces simples ocultam implementações complexas), reuso (com consequente aumento de produtividade no desenvolvimento), distribuição (permitindo implementar alta disponibilidade) e interoperabilidade (operação em ambientes heterogêneos).

2.1.2 Web Services

Segundo o *World Wide Web Consortium* (W3C, 2004, tradução nossa):

> O *Web Service* é uma aplicação de *software* identificada por uma URI, onde interfaces são definidas, descritas e publicadas como artefatos XML. Um *Web Service* permite interações diretas com outros agentes de *software* por meio da troca de mensagens XML, trafegadas sobre protocolos baseados em *internet*.

Os *Web Services* são aplicações que seguem um conjunto de padrões definidos pela *World Wide Web Consortium*, tais como XML, SOAP, WSDL, HTTP, e outros (W3C, 2001, 2004, 2007).

As principais características dos *Web Services* são: interoperabilidade (neutralidade de ambiente e linguagem), baixo acoplamento, padrões abertos e amplamente reconhecidos, baixa complexidade e facilidade de acesso (disponibilidade). Essas características tornam os *Web Services* uma excelente escolha para compor a implementação de uma Arquitetura Orientada a Serviços (SOA).

Além disso, há alguns padrões definidos para lidar com a segurança da informação em sistemas baseados em *Web Services*, garantindo aspectos como confidencialidade, autenticidade, integridade e não-repúdio, tais como *Secure Socket Layer* (SSL), *XML Signature*, *XML Encryption*, *Security Assertion Markup Language* (SAML), *WS-Security*, *WS-Policy*, *WS-Trust*, *WS-Federation*, e outros (WINDLEY, 2005).

2.1.3 Serviços REST

Representational State Transfer (REST) é um estilo de arquitetura de *software* para sistemas distribuídos através de protocolos *internet*. Tema abordado primariamente por Roy Fielding em sua tese de doutorado (FIELDING, 2000), o REST é utilizado como uma forma de oferecer serviços via protocolo HTTP (ou HTTPS, se usado em conjunto com SSL). É uma alternativa simplificada, leve e menos robusta do que os *Web Services*, pois os serviços REST não necessitam de

envelopes SOAP para a troca de mensagens - nem mesmo o XML é requerido; um artefato *Javascript Object Notation* (JSON), ou mesmo texto puro podem ser usados em seu lugar.

2.1.4 Barramento ESB (*Enterprise Service Bus*)

O *Enterprise Service Bus* (ESB) é uma camada de *software* (conhecida como *middleware*), que visa oferecer uma infraestrutura flexível de conectividade para a integração de serviços e aplicações (CHAPPELL, 2004). Numa arquitetura SOA, o ESB facilita a publicação e descoberta dos serviços disponíveis.

Para que as aplicações possam usufruir os benefícios de um ESB, existem elementos de *software* chamados conectores e adaptadores. Esses elementos possibilitam a integração entre os mais diversos sistemas, inclusive os legados (como aplicações em *mainframe* ou as desenvolvidas com linguagens de programação obsoletas).

Uma das características do ESB que simplificam a arquitetura SOA é também um de seus pontos mais críticos: a centralização. Como todos os serviços são conectados ao barramento, uma possível falha de *hardware* ou *software* pode comprometer o funcionamento de diversos serviços simultaneamente. Para minimizar esse tipo de problema, são aplicados conceitos como infraestrutura em *cluster* e computação em nuvem (LINTHICUM, 2010).

2.1.5 Computação em Nuvem

A computação em nuvem (*cloud computing*) é uma forma de computação distribuída, onde aplicações de *software* são executadas em diferentes servidores fisicamente separados, mas logicamente reunidas como um serviço ("*as a service*"). Uma das funções da computação em nuvem é oferecer infraestrutura de *hardware*, sistema básico, aplicação ou armazenamento, como serviço reutilizável, escalável, com alta disponibilidade e de custo menor se comparado ao de manter toda essa infraestrutura localmente (VELTE, 2010).

A característica de reunir capacidades e oferecê-las como serviços, faz da computação em nuvem uma ideia convergente com o conceito de SOA (LINTHICUM, 2010). Por esse motivo, em algumas implementações de Arquitetura Orientada a Serviços, a computação em nuvem é utilizada como parte da solução, provendo serviços de infraestrutura, armazenamento, e de plataforma para os serviços de negócio.

2.1.6 Criptografia em Sistemas Computacionais

Segundo o dicionário da língua portuguesa (FERREIRA, 1999), “criptografia é uma escrita secreta, em cifra, isto é, por meio de abreviaturas ou sinais convencionais”.

A criptografia em sistemas computacionais consiste da aplicação de algoritmos matemáticos e de uma chave criptográfica, de forma a tornar uma mensagem digital ilegível. Somente o destinatário da mensagem poderá recuperá-la (torná-la legível novamente), através do mesmo algoritmo e da chave criptográfica correspondente. Esse processo garante a privacidade da informação, já que somente o possuidor da chave poderá desembaralhar a mensagem. Qualquer outro indivíduo que intercepte essa mensagem, não poderá interpretá-la (a menos que utilize um procedimento denominado “quebra de chave de criptografia”, que consiste em usar poder de processamento para descobrir a chave por método de “força bruta”). O que garante a segurança da informação é o custo dessa quebra: quanto maior em relação ao valor da própria informação criptografada, menor será a possibilidade de que alguém tenha interesse em investir muito para obter um retorno comparativamente menor (SILVA et al., 2008, p. 13).

A criptografia em sistemas computacionais está dividida em três tipos de algoritmos (SILVA et al., 2008, p. 14; VOLPI, 2001, p. 8-16): algoritmos de resumo (*digest algorithms*), que transformam uma mensagem plana em uma chave de tamanho fixo (chamada de *hash*); algoritmos de chave secreta (criptografia simétrica), que usam uma chave para cifrar uma mensagem e a mesma chave para tornar a mensagem legível novamente; e algoritmos de chave pública (criptografia assimétrica), que utilizam um par de chaves, uma privada e outra pública, através

dos quais uma mensagem cifrada com uma das chaves só pode ser revelada por meio da outra chave do mesmo par.

2.1.7 Assinatura e Certificado Digital

Segundo Volpi (2001, p. 5), assinatura digital é “um mecanismo digital utilizado para fornecer confiabilidade, tanto sobre autenticidade de um determinado documento eletrônico, como sobre o remetente do mesmo”.

A assinatura digital é uma aplicação para a criptografia computacional, onde uma determinada mensagem (comumente chamada de “documento eletrônico”), é anexada a sua assinatura cifrada com uma chave privada. Essa assinatura é um resumo (*hash*) da mensagem original. O destinatário recebe o documento digitalmente assinado, aplica o algoritmo de resumo correspondente, e usa a chave pública do remetente para desembaralhar a assinatura digital recebida. Ao comparar os dois resumos (recebido e calculado), o destinatário tem condições de concluir sobre a autenticidade do documento (se ele foi assinado de fato pelo remetente) e também sobre sua integridade (se ninguém interceptou e alterou o conteúdo do documento durante o envio).

O certificado digital é um arquivo que contém uma chave pública, além de alguns atributos relacionados a entidade (indivíduo ou organização) titular dessa chave, associado a uma assinatura digital emitida por uma autoridade certificadora reconhecida. O certificado garante que o dono do par de chaves criptográficas é realmente quem ele afirma ser (SILVA et al., 2008, p. 26).

Assim como um cartório reconhece e autentica um documento em papel, uma autoridade certificadora confiável assina um certificado digital, após uma criteriosa análise de identidade do titular do mesmo. Essa é a condição que garante a autenticidade do certificado: confiança na autoridade certificadora. Por esse motivo, as autoridades certificadoras seguem rigorosas condutas de segurança e têm procedimentos rígidos para a emissão de certificados, incluindo datas de validade (expiração), listas de certificados revogados, cadastro dos dados dos titulares, e outros controles físicos e lógicos (LUZ, 2008).

2.1.8 Cartões Inteligentes (*Smart Cards*)

O cartão inteligente (mais conhecido como *smart card*) é um cartão de plástico, com dimensões semelhantes a um cartão magnético comum, porém dotado de um *chip* (circuito integrado) composto por microprocessador, memória e em muitos casos, um módulo interno com capacidade de processamento criptográfico (RANKL, 2003).

O *smart card* tem uma série de contatos dourados (conectores), através dos quais é realizada a comunicação a partir de um aparelho leitor compatível. Entretanto, há também o cartão sem contato (conhecido como *contactless*), que possui uma antena interna para comunicação sem fio com o aparelho leitor (muito usado em bilhetagem no transporte público, em grandes cidades brasileiras).

Outra modalidade de *smart card* é chamada de "*chip GSM*" (ou *SIM card*), possuindo dimensões menores e aplicado em aparelhos de telefonia móvel celular.

Ainda há o *token*, que é um dispositivo semelhante a uma chave (ou *pendrive*), é composto por um *chip* da mesma categoria do *smart card* - porém sem o cartão de plástico - e pode ser usado somente em computadores, através da conexão com uma de suas portas disponíveis.

Por sua capacidade de processamento e armazenamento (e pela presença de um módulo criptográfico interno), tanto o *smart card* como o *token* são dispositivos que oferecem características de segurança, praticidade e robustez necessárias ao armazenamento de dados sensíveis, chaves criptográficas e certificados digitais (RANKL, 2003).

2.2 Cenário Atual e Iniciativas Existentes

A Medida Provisória 2.200-2 de 24 de agosto de 2001 deu início à implantação do Sistema Nacional de Certificação Digital da ICP-Brasil (BRASIL, 2001). Segundo definição no portal do Instituto Nacional de Tecnologia da Informação – ITI (BRASIL, 2011):

> A Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil) é uma cadeia hierárquica e de confiança que viabiliza a emissão de certificados digitais para identificação do cidadão quando transacionando no meio virtual, como a *Internet*.

A partir desse momento, a certificação e a assinatura digital passaram a ser viáveis ao cidadão brasileiro, possibilitando o surgimento de algumas iniciativas de identificação digital: o e-CPF e o e-CNPJ, emitidos pela Receita Federal do Brasil; o Programa Universidade para Todos (ProUni), oferecendo acesso a seu sistema através de certificados digitais; a Agência Nacional de Saúde Suplementar (ANS), utilizando os certificados para troca de informações entre operadoras de planos de saúde; o Instituto Nacional de Propriedade Industrial (INPI), no seu sistema e-Marcas, para solicitação de registro de marcas e patentes; e a assinatura digital em diversos setores - como no Poder Judiciário, na Nota Fiscal Eletrônica (NF-e), no Sistema Público de Escrituração Digital (SPED) - dentre outros.

Além dessas iniciativas, o Registro de Identidade Civil (RIC) é um projeto que merece destaque, pois tem como meta a identificação única de todos cidadãos brasileiros por meio de um cadastro nacional, usando certificados digitais e cartões *smart card*.

A seguir, serão abordadas com mais detalhes as iniciativas do e-CPF, e-CNPJ e do RIC, pois elas tratam da emissão de documentos que identificam digitalmente os cidadãos, fundamento sobre o qual será construída a prova de conceito proposta neste trabalho.

2.2.1 CPF e CNPJ Eletrônico (e-CPF e e-CNPJ)

O e-CPF é uma versão eletrônica do Cadastro de Pessoa Física (CPF), que garante a autenticidade e a integridade nas transações eletrônicas de pessoas físicas. De modo semelhante, o e-CNPJ é uma versão eletrônica do Cadastro Nacional de Pessoa Jurídica (CNPJ), usado por empresas e outras entidades (BRASIL, 1999; CERTSIGN, 2011).

Figura 1 - Cartões do e-CPF e do e-CNPJ
Fonte: Receita Federal do Brasil (BRASIL, 2011).

Cada um desses documentos é composto por um certificado digital, que contém além do par de chaves (pública e privada) e da assinatura digital de uma autoridade confiável, os dados correspondentes aos diversos documentos do titular - como Registro Geral (RG), Título Eleitoral, número de inscrição no Programa de Integração Social (PIS), CPF ou CNPJ, dentre outros.

Existem dois tipos de certificado emitidos como e-CPF e e-CNPJ: o tipo A1 e o tipo A3. Os certificados A1, são válidos apenas por um ano, e são fornecidos como um simples arquivo protegido por uma senha. Já os certificados A3 têm validade de três anos, e são armazenados em dispositivos criptográficos (*tokens* ou *smart cards*), protegidos por uma senha pessoal, o *Personal Identification Number* (PIN).

Atualmente, os certificados e-CPF e e-CNPJ podem ser usados como forma de autenticação em serviços oferecidos pela Receita Federal do Brasil (RFB), como envio de declarações de imposto de renda e consulta de contribuintes em malha (BRASIL, 2011).

2.2.2 Registro de Identidade Civil (RIC)

O Registro de Identidade Civil (RIC) é uma inciativa do Ministério da Justiça que tem por objetivo unificar a identificação dos cidadãos brasileiros em um cadastro de nível nacional, prevendo a utilização de um cartão (*smart card*) para reunir todas as informações sobre o cidadão (BRASIL, 2010).

Figura 2 - Cartão do Registro de Identidade Civil (RIC)
Fonte: Portal do RIC, Ministério da Justiça (BRASIL, 2010).

Ainda em fase piloto e de experimentação, o cartão do RIC contém um certificado digital emitido por uma autoridade confiável, números de documentos (como o próprio número RIC, CPF, RG, Título Eleitoral, PIS, etc) e outros dados, todos gravados dentro do *chip* presente no cartão.

Além disso, o RIC possui mecanismos de segurança em sua impressão: fotografia e impressão digital do cidadão, marcas d'água, holografias, imagens aparentes sob luz ultravioleta, relevo, impressão a *laser* (sem uso de tintas), etc.

Todas essas características tornam o RIC um documento confiável e seguro, suficiente para identificar um cidadão, minimizando sobremaneira a possibilidade de fraudes.

Esse é um projeto ambicioso, pois abrange um cadastro de nível nacional, utiliza novas tecnologias que têm um custo maior do que os documentos tradicionais em papel e vai alterar drasticamente a maneira como um cidadão brasileiro é identificado.

2.3 Proposta de Integração de Sistemas

A proposta deste trabalho é apresentar uma arquitetura de integração de sistemas governamentais, oferecendo serviços ao cidadão, baseada na utilização de um cartão como o RIC para autenticar com segurança o usuário. Para atingir esse objetivo, duas abordagens serão exploradas: em uma delas, o cidadão continua utilizando vários sistemas, cada um com o seu propósito específico, porém compartilhando dados entre si, e unificando a autenticação através do mesmo cartão. Na outra abordagem, o cidadão acessa um único portal, que centraliza e agrega vários serviços. Em seguida, será apresentada uma proposta de integração de serviços privados e governamentais, de forma a tornar o cartão único de identificação (como o RIC), um meio seguro e amplamente aceito para autenticar um cidadão nos mais diferentes sistemas.

2.3.1 Sistemas Independentes Compartilhando Serviços

Um exemplo de arquitetura de integração de serviços ao cidadão, baseado no acesso a diferentes sistemas – porém interconectados entre si – é ilustrado no diagrama a seguir.

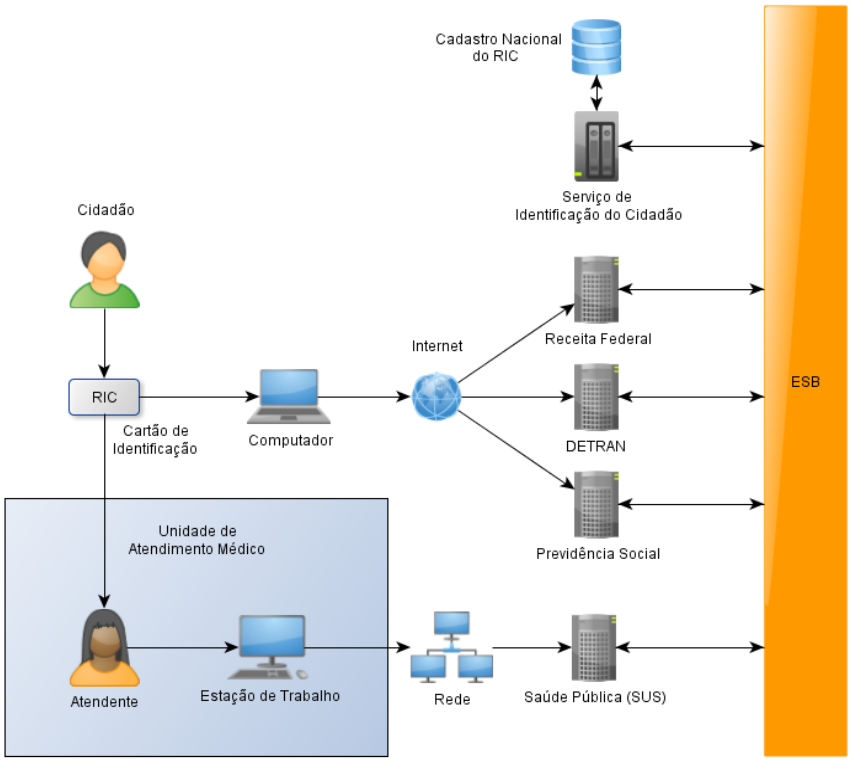

Figura 3 - Exemplo de integração com sistemas independentes

Neste exemplo, são disponibilizados ao cidadão, por meio de acesso via *internet*, alguns sistemas, tais como: Receita Federal – consulta de situação cadastral, de restituição de imposto de renda; Departamento de Trânsito (DETRAN) – consulta e recurso de multas, documentação veicular; Previdência Social – consulta a abonos sociais e contribuição oficial, obtenção de benefícios.

O cidadão, então, pode efetuar a sua autenticação em qualquer um desses sistemas, utilizando seu cartão único de identificação (RIC), através de um leitor de *smart cards* instalado em seu computador, sendo somente necessário digitar a senha que desbloqueia o certificado digital presente no *chip* do cartão (PIN).

Além disso, o cidadão pode utilizar seu cartão ao se apresentar numa unidade de atendimento médico, onde um atendente acessa um sistema gerencial de informações de saúde pública, conveniado com o Sistema Único de Saúde (SUS), por exemplo. Como ilustrado no diagrama, esse sistema poderia ser de uso exclusivo pelas unidades de atendimento, não disponível ao cidadão comum via *internet* – porém, como os outros sistemas citados anteriormente, ele também se beneficiaria da integração.

Nesta abordagem, cada sistema governamental tem sua própria base de dados, regras de negócio e interface com o usuário, exatamente como acontece atualmente. A novidade é a promoção desses sistemas como serviços, disponibilizando as informações de seu domínio para outros sistemas (serviços), através de tecnologias como *Web Services*, Barramento (ESB), adaptadores e conectores, etc. Essas tecnologias são largamente utilizadas na Arquitetura Orientada a Serviços (SOA), que preconiza que cada serviço deve disponibilizar as informações relevantes ao negócio, consequentemente facilitando o reuso.

Desta forma, cada serviço tem uma responsabilidade definida, e acessa outro serviço quando tem necessidade de obter dados, evitando a duplicação de informações e de processamento. Neste exemplo de integração, um dos serviços é responsável por autenticar um usuário a partir do certificado digital do seu cartão, validando sua identidade com base na autoridade certificadora emissora, datas de emissão/expiração e em um cadastro nacional. Todos os outros sistemas acessam esse serviço para realizar essa etapa de autenticação.

Do mesmo modo, um sistema da Previdência Social poderia acessar um serviço do SUS para obter informações de saúde sobre um determinado cidadão, com o objetivo de validar uma solicitação de um benefício motivado por afastamento médico.

A maior dificuldade de implementação desta abordagem está no fato de muitos dos sistemas hoje existentes possuírem bases de dados com informações redundantes, por exemplo: o sistema da Previdência possui todos os dados pessoais do cidadão – como nome, endereço, data de nascimento, etc., o sistema da Receita Federal também possui exatamente os mesmos dados; e assim por diante. Consequentemente, para que esta arquitetura seja implementada com sucesso, é vital que haja uma limpeza das aplicações existentes, removendo dados redundantes, e confiando de forma restrita, a cada serviço, o fornecimento dos dados de sua responsabilidade.

Alguns conceitos poderiam ser aplicados para auxiliar nessa questão, como d*ataware house e data mining* (depósito e mineração de dados). O principal propósito do processo de mineração de dados é extrair informações a partir de um conjunto de dados e transformá-las em uma estrutura compreensível para uso posterior (ACM-SIGKDD, 2006, tradução nossa).

Outra questão importante é a segurança da informação: é necessário que a implementação desta arquitetura aplique controles e políticas de segurança com o objetivo de evitar vazamento de informações, ou indisponibilidade de serviços, como criptografia, certificados de autenticação de serviços parceiros, infraestrutura em *cluster* ou nuvem, etc.

2.3.2 Portal do Cidadão Brasileiro

Nesta abordagem, o cidadão acessa os serviços disponibilizados através de um portal na *internet*, como demonstrado no diagrama a seguir.

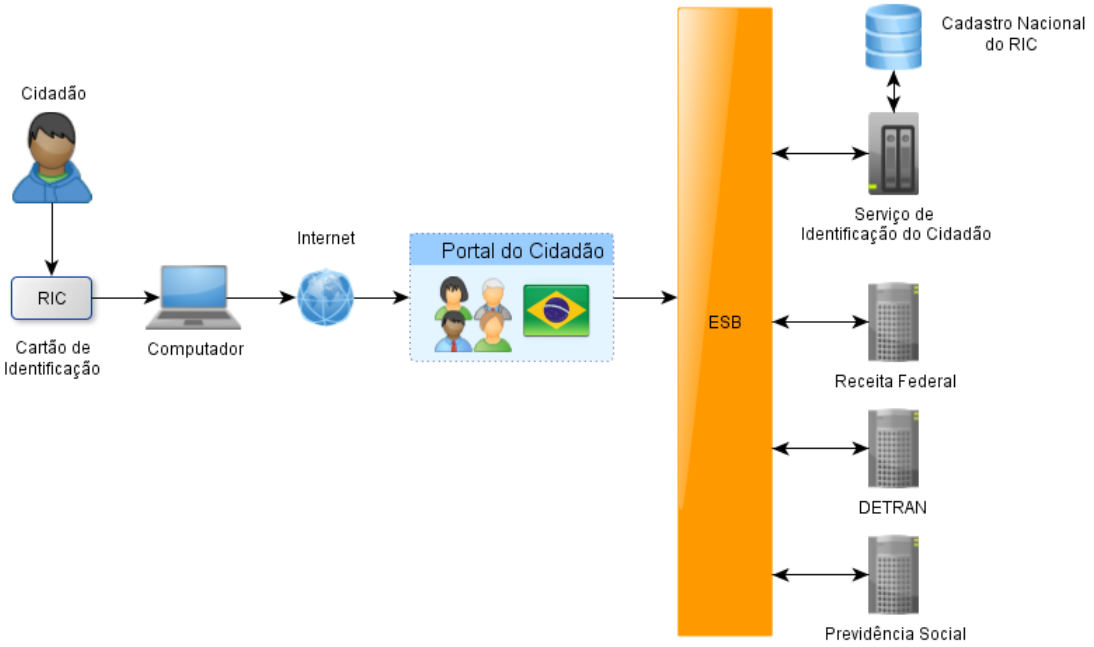

Figura 4 - Portal do Cidadão Brasileiro

Todos os sistemas atuais são modificados de forma a disponibilizar suas informações como serviços, interligados entre si, mas sem qualquer interface com o cidadão (usuário). O Portal passa a ser a interface única e centralizada de acesso aos serviços oferecidos pelos órgãos governamentais.

Neste exemplo, o cidadão realiza sua autenticação no Portal através do cartão de identificação (como o RIC) e acessa todos os serviços de forma prática, através de um menu de opções, presente na interface. Serviços oferecidos pela Receita Federal, DETRAN, Previdência Social, etc., estariam disponíveis desta maneira.

A maior vantagem desta abordagem é a facilidade de uso pelo cidadão, já que os serviços de que ele necessita seriam encontrados no mesmo lugar – o Portal do Cidadão Brasileiro.

Entretanto há alguns desafios de implementação e implantação deste tipo de arquitetura. Um deles é o grande esforço necessário para alterar os sistemas existentes e construir um novo - o Portal do Cidadão. Outro é característico da própria arquitetura: o Portal do Cidadão centralizaria muitos acessos de cidadãos interessados em interagir com diversos sistemas ao mesmo tempo. Isso tornaria o Portal um ponto crítico, suscetível a problemas de segurança e de desempenho, os quais poderiam causar a indisponibilidade dos serviços.

Tendo em vista esses aspectos, para que esta abordagem possa ser implantada com sucesso, é necessário considerar políticas de segurança, infraestrutura em *cluster* ou nuvem para promover alta disponibilidade, e tomar outras ações que visem a manutenção da qualidade de operação dos serviços oferecidos.

2.3.3 Integração de Sistemas Privados e Governamentais

Este é um exemplo de como empresas privadas também poderiam se beneficiar de uma integração com serviços governamentais, oferecendo facilidade de acesso ao seu cliente.

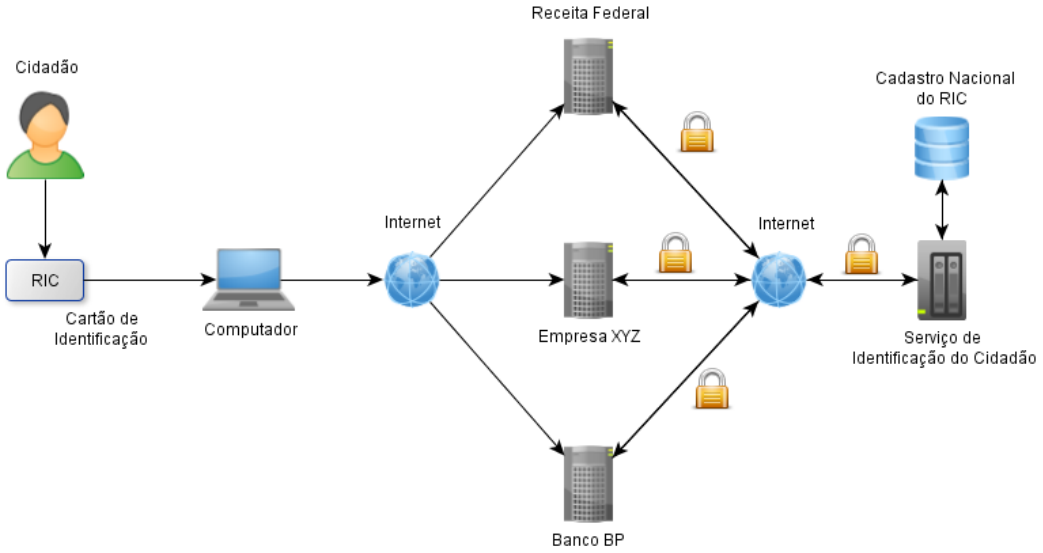

Figura 5 - Exemplo de integração entre sistemas privados e governamentais

Neste exemplo, o cidadão acessa um sistema privado de uma Empresa XYZ com o seu cartão de identificação, ou realiza uma transação bancária com o Banco BP, ou ainda, acessa algum serviço oferecido pela Receita Federal. Como ilustração, o cidadão poderia realizar uma compra via *e-commerce*, na loja virtual da Empresa XYZ. No momento do pagamento, o seu cartão de identificação seria usado para autenticar a transação com o Banco BP, que por sua vez, utilizaria um serviço oferecido pela Receita Federal para efetivar automaticamente a arrecadação de um imposto obrigatório nessa compra.

Nesta arquitetura, cada empresa ou órgão de governo disponibiliza serviços através da *internet*. Esses serviços podem ser construídos a partir de tecnologias como *Web Services* ou REST, e sua comunicação pode ser realizada através de algo como uma rede virtual privada (VPN), criptografada, autenticada a partir de certificados digitais, disponível apenas aos parceiros conhecidos dos serviços. Desta maneira, somente empresas conveniadas receberiam o acesso aos serviços governamentais mais restritos, evitando o uso indevido por qualquer intruso presente na *internet*.

A prova de conceito desenvolvida neste trabalho é baseada em uma abordagem semelhante a esta.

3 ENGENHARIA DE SOFTWARE

Esta seção contém os artefatos de documentação gerados durante a execução do projeto de desenvolvimento do protótipo (prova de conceito), o qual visa demonstrar os conceitos teóricos explorados nesta pesquisa.

3.1 Documento de Visão (DV)

3.1.1 Objetivo do Projeto

Com o objetivo de demonstrar um cenário de integração entre sistemas governamentais – e também privados – este trabalho contempla a implementação de uma prova de conceito, composta por alguns serviços integrados, oferecidos aos cidadãos na forma de aplicações *web*.

Para utilizar esses serviços, é necessário que o cidadão possua tão somente um cartão de identificação único, contendo um certificado digital emitido por uma autoridade certificadora reconhecida.

3.1.2 Partes Envolvidas e Usuários

(Sob o ponto de vista do trabalho acadêmico).

Nome	Descrição	Responsabilidades
Robson Martins (Aluno)	Líder de Projeto Analista Patrocinador	Gerenciar as atividades do projeto. Participar no desenvolvimento do produto. Avaliar custos e aprovar orçamentos. Analisar riscos e adequar cronogramas.
Matheus Haddad (Orientador)	Consultor Proprietário do Produto	Orientar sobre o desenvolvimento do produto e condução do projeto. Realizar testes de aceitação e avaliação final do produto.
Eduardo Endo (Coordenador)	Consultor	Orientar sobre questões gerais do projeto. Suportar questões institucionais.

Quadro 1 - Partes envolvidas e usuários

3.1.3 Posicionamento / Visão da Situação Atual

Atualmente, o cidadão necessita acessar diferentes sistemas para exercer suas atividades, e em cada um desses sistemas (governamentais ou privados), é requerido um diferente conjunto de senhas e nomes de usuário. Sendo assim, a identificação do cidadão em cada sistema é realizada de maneira desintegrada e descentralizada, exigindo dele um esforço adicional para memorizar ou anotar várias senhas, números e nomes de usuário.

3.1.4 Visão Geral da Solução Proposta

Para demonstrar a utilização de um cartão único de identificação como forma de autenticação e autorização entre diferentes sistemas integrados, é proposta a implementação de uma prova de conceito, da seguinte maneira:

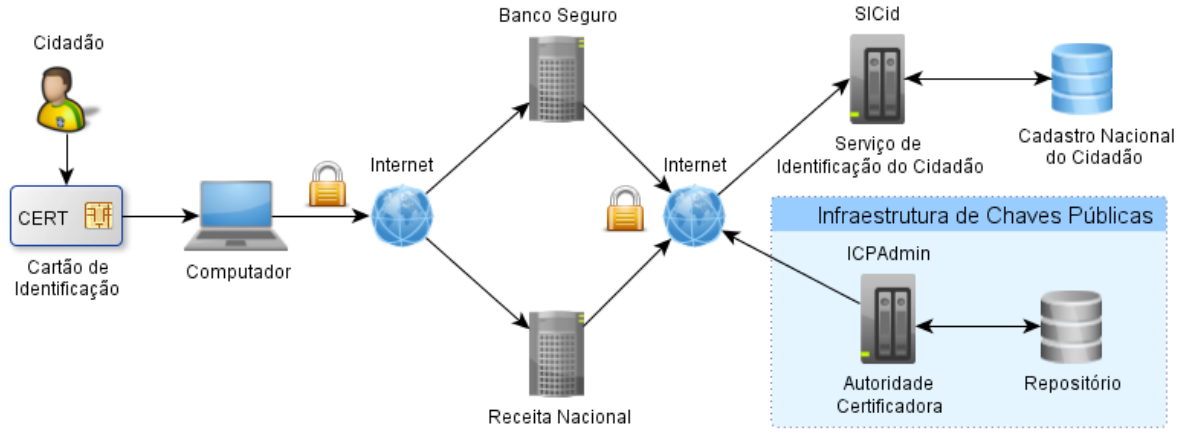

Figura 6 - Visão geral da solução proposta

Nesta solução, um cidadão pode acessar dois serviços, usando seu cartão de identificação: um privado - que é o Banco Seguro, e outro governamental - a Receita Nacional. Esses dois serviços, por sua vez, são integrados a um terceiro – também governamental: o Serviço de Identificação do Cidadão (SICid). Além disso, há uma Infraestrutura de Chaves Públicas (ICP) implementada, também integrada ao SICid.

A seguir são detalhados cada um desses serviços.

- **Serviço de Identificação do Cidadão (SICid)**

O Serviço de Identificação do Cidadão (SICid) é um serviço governamental que mantém o Cadastro Nacional do Cidadão. Todos os cidadãos são cadastrados com seus dados pessoais (nome, filiação, data de nascimento, números de documentos, etc.) e associados a certificados emitidos por uma Autoridade Certificadora (AC) reconhecida (detalhes mais adiante, no tópico sobre a aplicação ICP Admin).

Esse serviço disponibiliza métodos para consulta de informações sobre cidadãos por meio de seus certificados digitais. Além disso, é oferecido um método para a validação de certificados quaisquer (verificação do emissor do certificado – retornando se ele foi emitido por uma AC pertencente a uma cadeia confiável; e se esse certificado não foi revogado ou teve sua data de validade expirada).

Somente aplicações (ou serviços) confiáveis (cujo certificado de aplicação esteja previamente cadastrado no SICid) podem consumir esse serviço, mediante autenticação, garantindo a segurança (privacidade) das informações.

Uma aplicação *web* (SICid Admin) permite a configuração do serviço SICid (disponível somente para usuários autorizados com perfil administrativo).

- **Banco Seguro**

O Banco Seguro é um exemplo de sistema privado: é uma aplicação de *internet banking*, que oferece serviços bancários (movimentação de contas) a seus clientes.

Nesta prova de conceito, o Banco Seguro está integrado ao SICid, utilizando seus serviços para autenticar seus clientes e para fornecer dados cadastrais durante a abertura de novas contas.

Além disso, o Banco Seguro oferece um serviço que pode ser integrado com outros sistemas, disponibilizando métodos para movimentação *online* de contas de seus clientes. Somente aplicações (ou serviços) confiáveis (cujo certificado de aplicação esteja previamente cadastrado no Banco Seguro) podem consumir esse serviço, mediante autenticação, garantindo a segurança (privacidade) das informações.

- **Receita Nacional**

A Receita Nacional é um exemplo de sistema governamental responsável pela arrecadação de tributos por parte dos cidadãos. A aplicação *web* permite ao cidadão consultar sua situação tributária e realizar o pagamento do imposto devido (quando for o caso).

O sistema da Receita Nacional é integrado ao SICid para autenticar cidadãos e representantes do governo (que são os administradores da Receita, responsáveis por calcular periodicamente o tributo dos cidadãos).

A Receita Nacional também é integrada ao serviço do Banco Seguro, oferecendo ao cidadão que possui uma conta no referido banco a facilidade de efetuar o pagamento de seu tributo totalmente *online*, sem precisar ir até a agência ou usar o *internet banking*.

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin)**

Como todas as aplicações e sistemas que compõem esta prova de conceito utilizam certificados digitais como vetor de autenticação de usuários e serviços, a existência de uma infraestrutura responsável pela emissão, revogação e renovação de certificados digitais se torna fortemente requerida; sendo assim, uma Infraestrutura de Chaves Públicas (ICP) deve ser acionada com esse propósito.

Devido ao alto custo e elevada burocracia de se utilizar uma Autoridade Certificadora (AC) publicamente reconhecida (como as da ICP-Brasil ou de outras autoridades internacionais) para a emissão de certificados de demonstração, a instituição de uma ICP própria (com certificado raiz autoassinado) é suficiente para esta prova de conceito.

Para tanto, a solução está na criação de uma aplicação para administrar o ciclo de vida de uma ICP, através de funções de emissão, revogação e renovação de certificados digitais. Essa aplicação é chamada de ICP Admin.

A aplicação *web* ICP Admin está integrada ao SICid, permitindo a validação de certificados dos seus usuários (que devem se autenticar com perfil administrativo).

3.1.5 Não Faz Parte do Escopo

Este projeto não visa explorar técnicas relacionadas à segurança da informação, embora alguns conceitos básicos estejam presentes (como criptografia, assinatura digital, autenticação, etc.).

Como o foco principal do projeto é demonstrar a integração, os serviços implementados são meros exemplos, construídos de maneira simplificada. Eles possuem apenas algumas funcionalidades básicas em suas regras de negócio, modelagem minimalista de banco de dados e interface com usuário sem muitos requintes visuais e de usabilidade.

Também não é escopo deste projeto lidar com questões relativas à disponibilidade dos serviços ou níveis de desempenho das aplicações. Como é uma prova de conceito, este projeto não leva em consideração nem quantidade de acessos simultâneos e nem tempos de resposta dos sistemas.

Por uma questão de limitação de tempo disponível para o desenvolvimento, conceitos como *dataware house*, *data mining* (mineração de dados), *open-gov* (governo aberto) e computação móvel (*mobile*) não foram considerados na implementação desta prova de conceito.

3.1.6 Funcionalidades

Esta prova de conceito contempla as principais funcionalidades, categorizadas por sistema:

- **[SIST.01] Serviço de Identificação do Cidadão (SICid)**

[FUNC.01.01] Funcionalidade – Manter o Cadastro Nacional do Cidadão.
Usuários com perfil administrativo (para o SICid) devem ser capazes de manter o Cadastro Nacional do Cidadão (adicionar, editar ou excluir cidadãos do cadastro).

[FUNC.01.02] Funcionalidade – Manter o cadastro da cadeia de certificados confiáveis.

Usuários com perfil administrativo (para o SICid) devem ser capazes de manter o cadastro da cadeia de certificados confiáveis (estabelecer a cadeia de certificados das Autoridades Certificadoras consideradas confiáveis para os sistemas que utilizam os serviços do SICid).

[FUNC.01.03] Funcionalidade – Manter o cadastro de aplicações confiáveis.

Usuários com perfil administrativo (para o SICid) devem ser capazes de manter o cadastro de aplicações confiáveis (adicionar ou excluir os certificados das aplicações que poderão consumir os serviços do SICid).

[FUNC.01.04] Funcionalidade – Validar certificado.

Sistemas externos com perfil de aplicação confiável (para o SICid) devem ser capazes de obter informações sobre a validação de um certificado digital qualquer.

[FUNC.01.05] Funcionalidade – Consultar cidadão.

Sistemas externos com perfil de aplicação confiável (para o SICid) devem ser capazes de obter dados cadastrais de um cidadão, a partir de seu certificado digital.

[FUNC.01.06] Funcionalidade – Listar cidadãos.

Sistemas externos com perfil de aplicação confiável (para o SICid) devem ser capazes de obter uma lista contendo dados cadastrais dos cidadãos constantes no Cadastro Nacional do Cidadão.

- **[SIST.02] Banco Seguro**

[FUNC.02.01] Funcionalidade – Efetuar saque na conta.

Usuários com perfil de cliente ou gerente (para o Banco Seguro) devem ser capazes de realizar saque de valores na sua conta (somente se possuir conta no banco).

Sistemas externos com perfil de aplicação confiável (para o Banco Seguro) devem ser capazes de realizar saque de valores na conta de um cliente, a partir de seu certificado digital, se ele possuir conta no banco.

[FUNC.02.02] Funcionalidade – Efetuar depósito em conta.

Usuários com perfil de cliente ou gerente (para o Banco Seguro) devem ser capazes de efetuar depósito de valores na sua conta (somente se possuir conta no banco).

Sistemas externos com perfil de aplicação confiável (para o Banco Seguro) devem ser capazes de efetuar depósito de valores na conta de um cliente, a partir de seu certificado digital, se ele possuir conta no banco.

[FUNC.02.03] Funcionalidade – Pagar títulos ou documentos.

Usuários com perfil de cliente ou gerente (para o Banco Seguro) devem ser capazes de pagar títulos (documentos) através de sua conta (somente se possuir conta no banco).

Sistemas externos com perfil de aplicação confiável (para o Banco Seguro) devem ser capazes de pagar títulos (documentos) através da conta de um cliente, a partir de seu certificado digital, se ele possuir conta no banco.

[FUNC.02.04] Funcionalidade – Transferir valores para outra conta.

Usuários com perfil de cliente ou gerente (para o Banco Seguro) devem ser capazes de efetuar transferência de valores para outra conta da mesma instituição bancária (somente se possuir conta no banco).

Sistemas externos com perfil de aplicação confiável (para o Banco Seguro) devem ser capazes de efetuar transferência de valores para outra conta da mesma instituição bancária, a partir do certificado digital de um cliente, se ele possuir conta no banco.

[FUNC.02.05] Funcionalidade – Consultar saldo e extrato.

Usuários com perfil de cliente ou gerente (para o Banco Seguro) devem ser capazes de consultar saldo ou extrato de sua conta (somente se possuir conta no banco).

[FUNC.02.06] Funcionalidade – Abrir nova conta.

Usuários com perfil de gerente (para o Banco Seguro) devem ser capazes de abrir contas para novos clientes do banco.

[FUNC.02.07] Funcionalidade – Fechar conta.
Usuários com perfil de gerente (para o Banco Seguro) devem ser capazes de fechar contas existentes.

[FUNC.02.08] Funcionalidade – Manter cadastro de aplicações confiáveis.
Usuários com perfil de gerente (para o Banco Seguro) devem ser capazes de manter o cadastro de aplicações confiáveis (adicionar ou excluir os certificados das aplicações que poderão consumir os serviços do Banco Seguro).

- **[SIST.03] Receita Nacional**

[FUNC.03.01] Funcionalidade – Consultar situação tributária.
Usuários com perfil de cidadão ou administrador - representante de governo (para a Receita Nacional) devem ser capazes de consultar sua situação tributária, ou seja, se têm pendências de imposto a pagar ou se estão regularizados.

[FUNC.03.02] Funcionalidade – Pagar tributo.
Usuários com perfil de cidadão ou administrador - representante de governo (para a Receita Nacional) devem ser capazes de pagar tributos, caso tenham pendências (imposto devido), online, através de uma conta no Banco Seguro (caso possuam uma).

[FUNC.03.03] Funcionalidade – Tributar cidadãos.
Usuários com perfil de administrador - representante de governo (para a Receita Nacional) devem ser capazes de calcular tributos e atribui-los aos cidadãos presentes no Cadastro Nacional do Cidadão.

3.1.7 Funcionalidades de Outros Sistemas

- **[SIST.04] Sistema ICP Admin**

[FUNC.04.01] Funcionalidade – Gerenciar emissão, revogação e renovação de certificados digitais.

Usuários com perfil administrativo (para o ICP Admin) devem ser capazes de gerenciar todo o ciclo de vida da ICP, instituída com o propósito de emitir certificados digitais para os servidores (computadores), aplicações (sistemas) e cidadãos (usuários) que compõem esta prova de conceito.

3.1.8 Atributos das Funcionalidades

Funcionalidade	Importância	Risco	Prioridade	Complexidade
[FUNC.01.01]	Essencial	Alto	Alta	Alta
[FUNC.01.02]	Secundária	Baixo	Alta	Média
[FUNC.01.03]	Secundária	Baixo	Alta	Média
[FUNC.01.04]	Secundária	Baixo	Alta	Média
[FUNC.01.05]	Essencial	Alto	Alta	Alta
[FUNC.01.06]	Desejável	Baixo	Média	Média
[FUNC.02.01]	Desejável	Baixo	Baixa	Baixa
[FUNC.02.02]	Secundária	Baixo	Média	Baixa
[FUNC.02.03]	Essencial	Médio	Alta	Baixa
[FUNC.02.04]	Desejável	Baixo	Baixa	Baixa
[FUNC.02.05]	Essencial	Alto	Alta	Baixa
[FUNC.02.06]	Essencial	Alto	Alta	Baixa
[FUNC.02.07]	Desejável	Baixo	Baixa	Baixa
[FUNC.02.08]	Secundária	Baixo	Alta	Média
[FUNC.03.01]	Essencial	Alto	Média	Baixa
[FUNC.03.02]	Essencial	Alto	Média	Média
[FUNC.03.03]	Secundária	Médio	Média	Baixa
[FUNC.04.01]	Essencial	Médio	Alta	Alta

Quadro 2 - Atributos das funcionalidades

Significado dos atributos:

Funcionalidade: Identificador da funcionalidade referida.

Importância: Importância da funcionalidade no ponto de vista das partes envolvidas no projeto. Pode assumir um dos valores:

• *Essencial*: Tarefas principais do sistema, suas funções básicas. Se forem omitidas, o sistema falhará na sua missão.

• *Secundário*: Englobam as funções de suporte ao sistema, tais como dados estatísticos, geração de relatórios, supervisão, e funções de teste. Se omitidos o sistema ainda assim (por um tempo) pode atingir sua missão principal mas, com qualidade degradada.

• *Desejável*: São características de “conforto”, não ligadas à missão principal do sistema, mas que ajudam na sua utilização.

Risco: Probabilidade de que a implementação da funcionalidade pode provocar eventos indesejáveis e significativos como, por exemplo, estouro nos prazos, ultrapassar o orçamento ou cancelamento. Pode assumir um dos valores:

• *Alto*: maior que 50%.

• *Médio*: entre 10 e 50%.

• *Baixo*: menor que 10%.

Prioridade: Uma vez que o analista tem a lista de funcionalidades e o proprietário do produto definiu o grau de necessidade de cada uma, cabe ao analista e ao líder de projeto definir o nível de prioridade de cada funcionalidade:

• *Alta*: Requisitos de alta relevância para a aplicação. A não incorporação desta funcionalidade no sistema afeta a satisfação final do proprietário do produto.

• *Média*: Identifica funcionalidades que são úteis ao sistema, mas cuja ausência não compromete o funcionamento geral do aplicativo.

• *Baixa*: Funcionalidades que não comprometem o sucesso da aplicação junto ao proprietário do produto.

Complexidade: Nível de dificuldade de implementação da funcionalidade. Pode assumir um dos valores:

• *Alta*: A funcionalidade possui muitas características e é necessário um grande esforço da equipe para incorporação desta, com grande consumo de recursos técnicos e humanos.

• *Média*: Um esforço razoável deverá ser feito pela equipe, gerando um consumo médio de recursos. Implementação com um custo aceitável.

• *Baixa*: Funcionalidade simples, que requer um pequeno esforço da equipe, envolvendo um consumo não significativo de recursos.

3.1.9 Especificações Suplementares

Premissas

Todas as aplicações devem ser:

- Construídas usando tecnologias e padrões de arquitetura Java;
- Implantadas em servidores de aplicação *Java Enterprise Edition* (JEE), com compatibilidade mínima de Java 5 (sugerido *JBoss* AS 6.1 *Community*), rodando em Sistema Operacional Livre e de Código Aberto (sugerido *GNU/Linux*);
- Acessíveis via navegador *web*, independente do sistema operacional do computador do cliente (para navegadores e sistemas mais populares);
- Protegidas por criptografia na camada de transporte (usando SSL);
- Autenticadas (somente) através de certificado digital, emitido por uma autoridade certificadora confiável (instituída para esse fim), e armazenado no cartão único de identificação do cidadão (*smart card*).

3.2 Modelo Entidade-Relacionamento (MER)

Este documento apresenta a modelagem de dados dos sistemas que compõem a prova de conceito, cuja implementação segue as especificações constantes no artefato “Documento de Visão (DV)”, encontrado na seção 3.1.

3.2.1 Modelo Conceitual

- **Serviço de Identificação do Cidadão (SICid):**

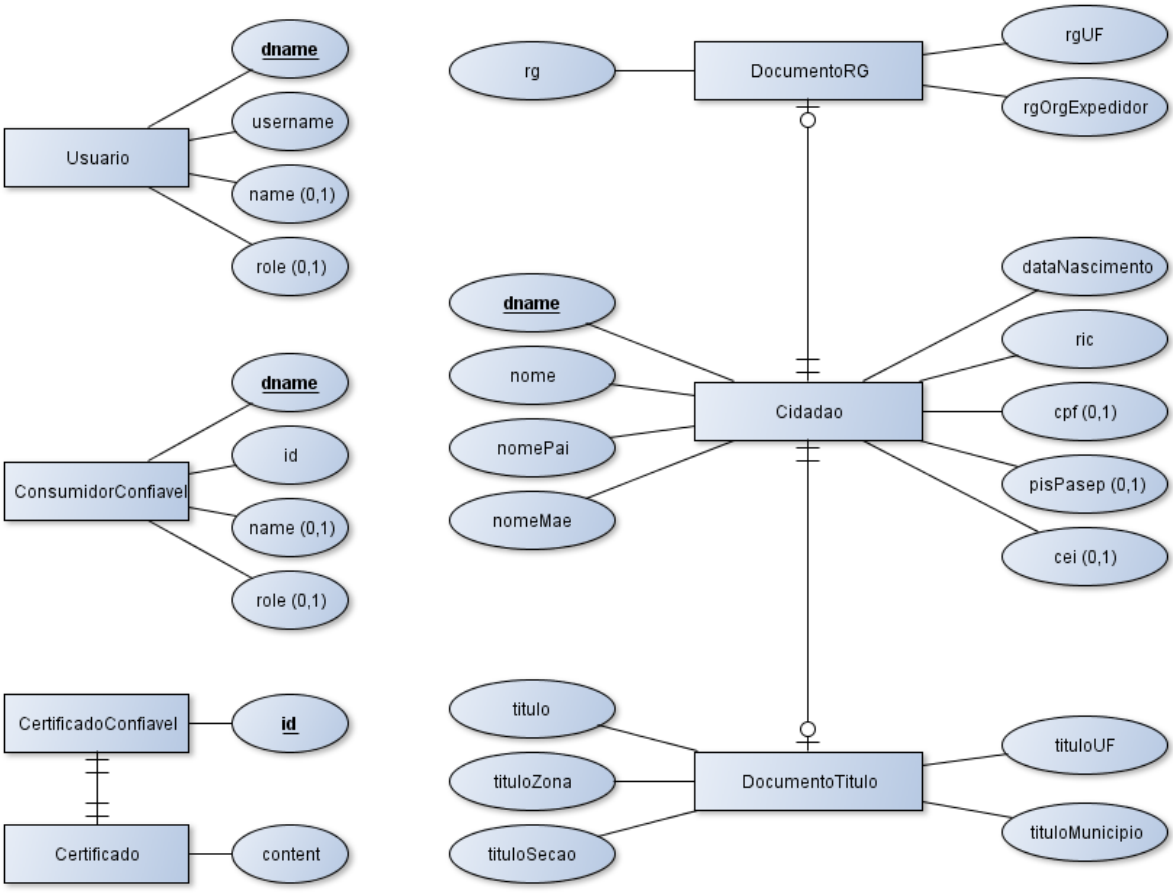

Figura 7 - Diagrama entidade-relacionamento do serviço SICid

Notas:

- A entidade **Usuario** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *username* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- A entidade **ConsumidorConfiavel** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *id* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- As entidades **CertificadoConfiavel** e **Certificado** estão modeladas separadas por questões de implementação: a entidade **Certificado** pode realizar a serialização de um objeto no padrão JCA (*X509Certificate*) para uma representação Base64 do conteúdo (*content*), e vice-versa.

- **Banco Seguro:**

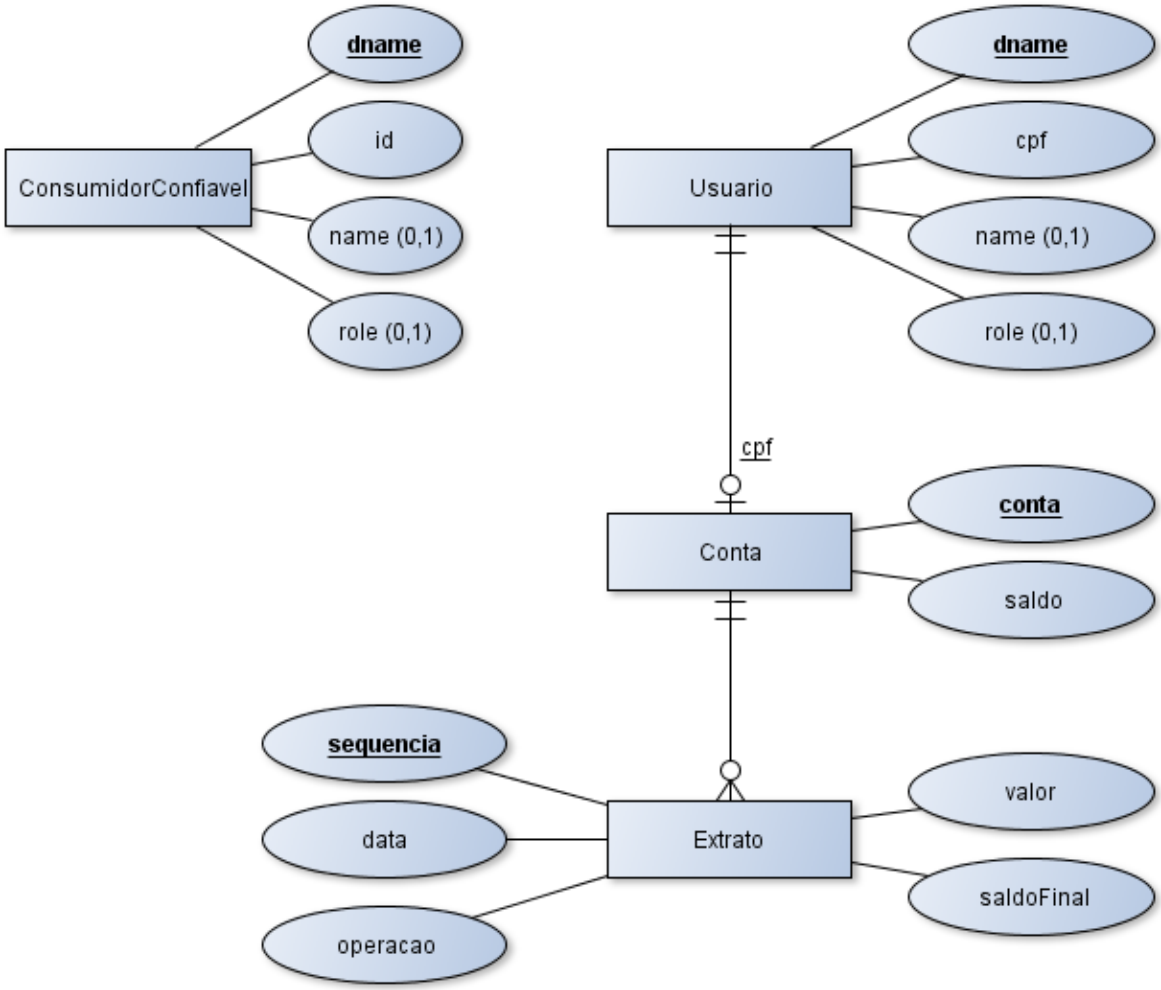

Figura 8 - Diagrama entidade-relacionamento do sistema Banco Seguro

Notas:

- A entidade **Usuario** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *cpf* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- A entidade **ConsumidorConfiavel** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *id* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- Uma **Conta** está associada a um CPF de **Usuario**, que pode ter, no máximo, somente uma conta (0,1).

- **Receita Nacional:**

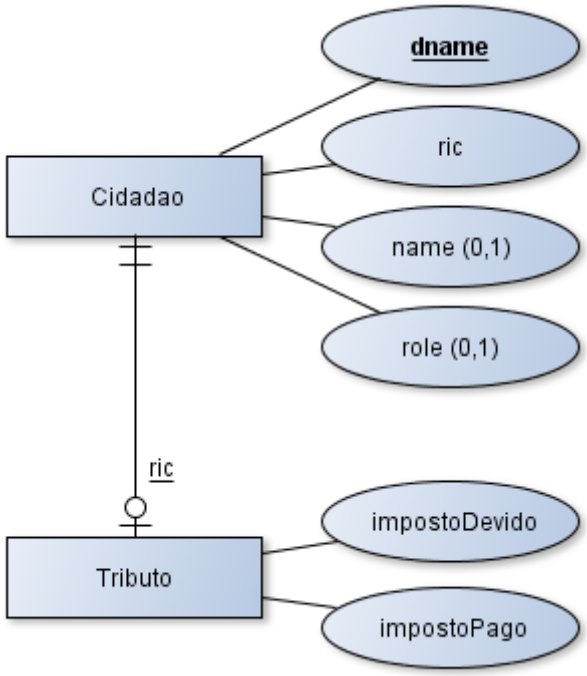

Figura 9 - Diagrama entidade-relacionamento do sistema Receita Nacional

Notas:

- A entidade **Cidadao** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *ric* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- Um **Tributo** está associado a um RIC de **Cidadao**, que pode ter, no máximo, somente um tributo (0,1).

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

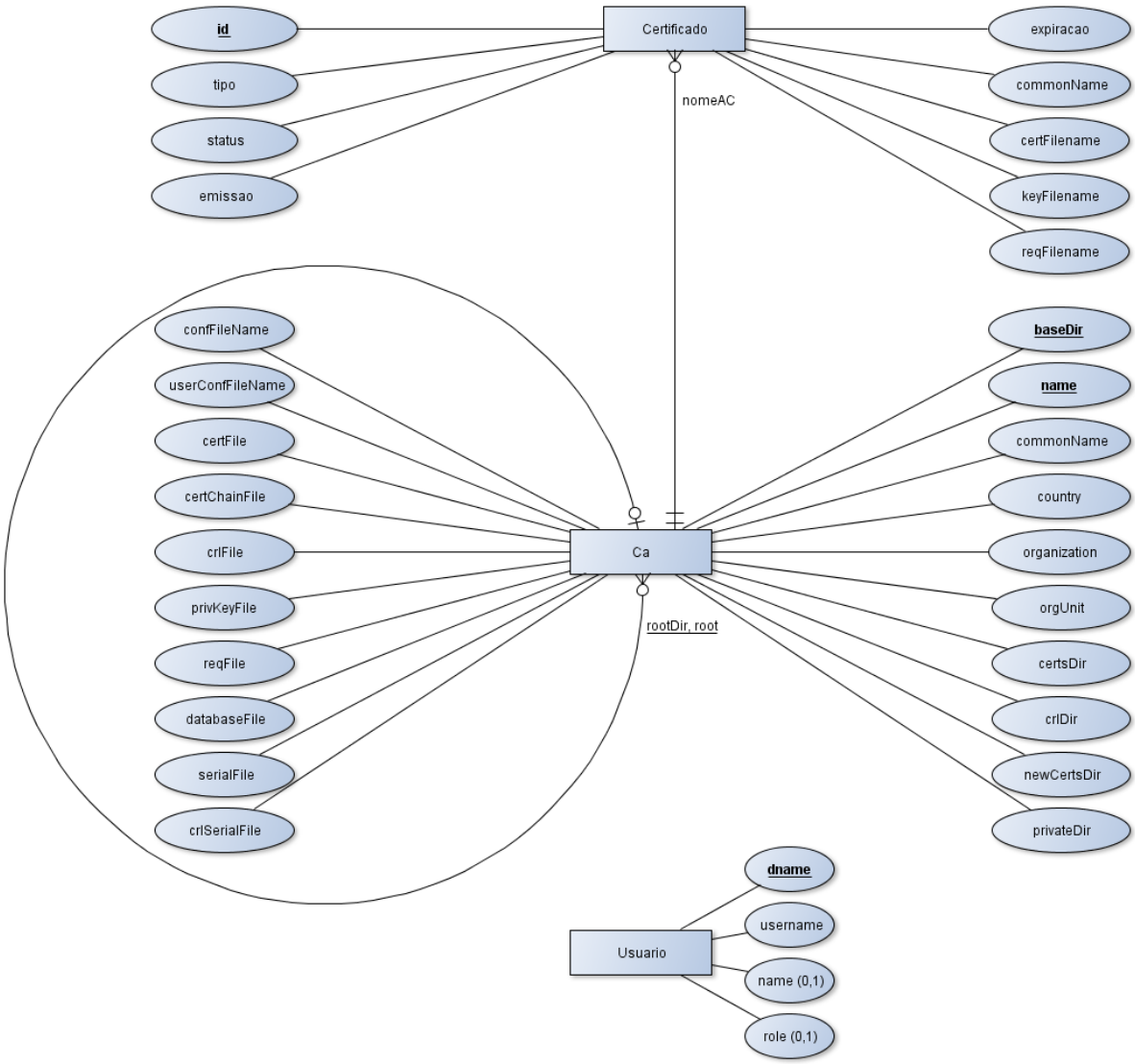

Figura 10 - Diagrama entidade-relacionamento do sistema ICP Admin

Notas:

- A entidade **Usuario** é usada para autenticação via *Login Module* no padrão JAAS, por este motivo ela implementa os atributos *username* (atribuído como *Principal Identity*) e *role* (atribuído como *Role*).
- A entidade **Ca** não é persistida em Banco de Dados, mas sim no sistema de arquivos, e é uma necessidade da implementação em torno do OpenSSL.
- Um **Certificado** está associado a uma **Ca** através do atributo *nomeAC*, que contém uma representação do caminho da AC no sistema de arquivos (caminho relativo, composto do diretório base e do nome da AC).

3.2.2 Modelo Lógico

- **Serviço de Identificação do Cidadão (SICid):**

Usuario: Representa um usuário do módulo administrativo.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado do usuário.
	username	texto			Nome (ID) de usuário.
	name	texto		X	Nome completo do usuário.
	role	texto		X	Perfil do usuário.
ConsumidorConfiavel: Representa uma aplicação consumidora dos serviços oferecidos.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado da aplicação.
	id	texto			Nome (ID) da aplicação.
	name	texto		X	Nome completo da aplicação.
	role	texto		X	Perfil da aplicação.
CertificadoConfiavel: Representa um certificado confiável na cadeia.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	id	inteiro	longo		Identificador numérico único.
	content	texto			Conteúdo do certificado, codificado em Base64.
Cidadao: Representa um cidadão cadastrado.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado do cidadão.
	nome	texto			Nome completo.
	nomePai	texto			Nome completo do Pai.
	nomeMae	texto			Nome completo da Mãe.
	dataNascimento	data			Data de Nascimento.
	ric	texto	11		Número do Registro de Identidade Civil (RIC).
	cpf	texto	11	X	Número do Cadastro de Pessoa Física (CPF).
	pisPasep	texto	11	X	Número de inscrição no Programa de Integração Social (PIS) ou Programa de Formação do Patrimônio do Servidor Publico (PASEP).

	cei	texto	12	X	Número do Cadastro Específico do INSS (CEI).
	rg	texto	15	X	Número do Registro Geral (RG).
	rgOrgExpedidor	texto	4	X	RG: Órgão Expedidor.
	rgUF	texto	2	X	RG: Sigla da Unidade da Federação (UF).
	titulo	texto	12	X	Número do Título Eleitoral.
	tituloZona	texto	3	X	Título: Zona eleitoral.
	tituloSecao	texto	4	X	Título: Seção eleitoral.
	tituloMunicipio	texto	20	X	Título: Município.
	tituloUF	texto	2	X	Título: Sigla da Unidade da Federação (UF).

Quadro 3 - Modelo lógico de dados do serviço SICid

- **Banco Seguro:**

Usuario: Representa um gerente ou cliente do banco.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado do usuário.
	cpf	texto			Número do Cadastro de Pessoa Física (CPF).
	name	texto		X	Nome completo do usuário.
	role	texto		X	Perfil do usuário.
ConsumidorConfiavel: Representa uma aplicação consumidora dos serviços oferecidos.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado da aplicação.
	id	texto			Nome (ID) da aplicação.
	name	texto		X	Nome completo da aplicação.
	role	texto		X	Perfil da aplicação.
Conta: Representa uma conta bancária.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	conta	inteiro	longo		Número único que identifica uma conta.
FK	cpf	texto			Número do Cadastro de Pessoa Física (CPF) do cliente associado.
	saldo	fracionário (real)	curto		Valor monetário que representa o saldo da conta.

Extrato: Representa uma entrada de extrato bancário.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	sequencia	inteiro	longo		Identificador numérico único.
FK	conta	inteiro	longo		Número que identifica a conta associada.
	data	data			Data da operação.
	operacao	inteiro enumeração			Código da operação bancária. Pode ser um desses: {DESCONHECIDA, TRANSFERENCIA, PAGAMENTO, SAQUE, DEPOSITO, DEPOSITO_TRANSF}
	valor	fracionário (real)	curto		Valor da operação.
	saldoFinal	fracionário (real)	curto		Saldo após a realização da operação bancária.

Quadro 4 - Modelo lógico de dados do sistema Banco Seguro

- **Receita Nacional:**

Cidadao: Representa um cidadão ou administrador (representante de governo).					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado do cidadão.
	ric	texto			Número do Registro de Identidade Civil (RIC).
	name	texto		X	Nome completo do cidadão.
	role	texto		X	Perfil do cidadão.
Tributo: Representa uma tributação associada a um cidadão.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P, FK	ric	texto			Número do Registro de Identidade Civil (RIC) do cidadão.
	impostoDevido	fracionário (real)	curto		Valor do imposto devido pelo cidadão.
	impostoPago	booleano			Indica se o cidadão já realizou o pagamento.

Quadro 5 - Modelo lógico de dados do sistema Receita Nacional

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

Usuario: Representa um usuário do módulo administrativo.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	dname	texto			*Distinguished Name* (DN) do certificado do usuário.
	username	texto			Nome (ID) de usuário.
	name	texto		X	Nome completo do usuário.
	role	texto		X	Perfil do usuário.
Certificado: Representa um certificado digital emitido por uma Autoridade Certificadora.					
Chave	**Atributo**	**Tipo**	**Tamanho**	**Opcional**	**Descrição**
P	id	inteiro	longo		Identificador numérico único.
	tipo	inteiro enumeração			Tipo de certificado. Pode ser um desses: {ECPF, ECNPJ, RIC, ECODIGO, ESERVIDOR, EAPLICACAO}
	nomeAC	texto			Nome da AC emissora (acrescido do caminho relativo).
	status	inteiro enumeração			Status do certificado. Pode ser um desses: {VALIDO, EXPIRADO, REVOGADO}
	emissao	data			Data de emissão do certificado.
	expiracao	data			Data de validade do certificado.
	commonName	texto			Common Name (CN) do certificado.
	certFilename	texto			Nome do arquivo de certificado (acrescido do caminho relativo da AC emissora).
	keyFilename	texto			Nome do arquivo de chave privada do certificado (acrescido do caminho relativo da AC emissora).
	reqFilename	texto			Nome do arquivo de requisição do certificado - PKCS#10 (acrescido do caminho relativo da AC emissora).

Ca: Representa uma Autoridade Certificadora. **Nota**: Esta entidade não é persistida em Banco de Dados.		
Atributo	**Tipo**	**Descrição**
baseDir	texto	Diretório base onde residem os dados das CA's (Autoridades Certificadoras).
name	texto	Nome único da CA (Autoridade Certificadora).
commonName	texto	Nome Comum (Common Name - CN) do certificado da CA (Autoridade Certificadora).
country	texto	País (Country - C) da CA (Autoridade Certificadora).
organization	texto	Organização (Organization - O) da CA (Autoridade Certificadora)
orgUnit	texto	Unidade Organizacional (Organisational Unit - OU) da CA (Autoridade Certificadora).
certsDir	texto	Nome do subdiretório de certificados gerados pela CA (Autoridade Certificadora).
crlDir	texto	Nome do subdiretório da Lista de Certificados Revogados (LCR) gerados pela CA (Autoridade Certificadora).
newCertsDir	texto	Nome do subdiretório de certificados de usuário gerados pela CA (Autoridade Certificadora).
privateDir	texto	Nome do subdiretório da chave privada da CA (Autoridade Certificadora).
confFileName	texto	Nome do arquivo de configuração (OpenSSL) da CA (Autoridade Certificadora).
userConfFileName	texto	Nome do arquivo de configuração (OpenSSL) de usuário da CA (Autoridade Certificadora).
certFile	texto	Nome do arquivo de certificado da CA (Autoridade Certificadora).
certChainFile	texto	Nome do arquivo da cadeia de certificados da CA (Autoridade Certificadora).
crlFile	texto	Nome do arquivo da Lista de Certificados Revogados (LCR) da CA (Autoridade Certificadora).
privKeyFile	texto	Nome do arquivo de chave privada da CA (Autoridade Certificadora).
reqFile	texto	Nome do arquivo de requisição da CA (Autoridade Certificadora).
databaseFile	texto	Nome do arquivo de *database* (*index*) da CA (Autoridade Certificadora).
serialFile	texto	Nome do arquivo de *serial* da CA (Autoridade Certificadora).
crlSerialFile	texto	Nome do arquivo de *serial* da Lista de Certificados Revogados (LCR) da CA (Autoridade Certificadora).
rootDir	texto	Diretório base da CA Raiz (issuer) desta CA (Autoridade Certificadora).
root	texto	Nome da CA Raiz (issuer) desta CA (Autoridade Certificadora).

Quadro 6 - Modelo lógico de dados do sistema ICP Admin

3.3 Documento de Arquitetura (DAQ)

Este documento apresenta a visão abrangente de arquitetura da prova de conceito, implementada conforme especificações constantes no artefato “Documento de Visão (DV)”, encontrado na seção 3.1 deste trabalho.

3.3.1 Metas e Restrições da Arquitetura

Esta prova de conceito demonstra um cenário de integração entre sistemas, tendo como uma de suas mais importantes premissas a autenticação integrada de um usuário através de seu cartão de identificação. Desta forma, algumas restrições e diretrizes fazem parte de sua especificação:

- **Utilização de certificados digitais como vetor de autenticação de usuários e sistemas**

Essa diretriz torna obrigatória a existência de uma infraestrutura responsável pela emissão, revogação e renovação de certificados digitais (Infraestrutura de Chaves Públicas – ICP). Para fins de demonstração, uma ICP própria (com certificado raiz autoassinado) é suficiente.

- **Armazenamento de par de chaves (e certificado digital associado) em mídia criptográfica apropriada**

Um usuário deve possuir um cartão de identificação, que armazena o certificado digital, ou seja, um cartão inteligente (*smart card*). Isso implica na necessidade de instalação, no computador do usuário, de *hardware* (aparelho leitor) e *software* (*driver* de dispositivo), destinados a ler essas informações contidas no *chip* do cartão e repassá-las ao módulo de autenticação das aplicações *web*.

Para fins de teste, uma exceção é a utilização de um arquivo protegido por senha, no formato PKCS#12 (RSA, 1999), contendo em seu interior o par de chaves criptográficas e o certificado digital correspondente.

- **Criptografia na camada de transporte (SSL sobre HTTP)**

Para garantir a confidencialidade das informações, na camada de transporte (DAVIE, PETERSON, 2007), é necessário que os dados que trafegam entre o navegador *web* do usuário e o servidor de aplicação sejam criptografados. Para tanto, é utilizado o protocolo *Secure Sockets Layer* (SSL) sobre o protocolo *Hypertext Transfer Protocol* (HTTP), resultando na combinação conhecida como HTTPS (HTTP seguro).

Esta diretriz implica na instalação de um par de chaves criptográficas para cada servidor de aplicação (e servidor *web*), cujo certificado tenha sido emitido por uma autoridade certificadora (AC) confiável ao usuário. No caso da ICP própria (com certificado raiz autoassinado), o usuário deverá instalar a cadeia de certificados da ICP em seu navegador (ou provisoriamente, num caso não-ideal, adicionar uma exceção de segurança), para que as aplicações possam ser acessadas via HTTPS.

- **Autenticação e criptografia nos serviços *web***

Para garantir confidencialidade, autenticidade e outros aspectos de segurança na troca de mensagens com os serviços *web*, é necessário que cada um deles seja implementado e configurado para usar criptografia via HTTPS, e além disso, seguindo um modelo de autenticação cujo vetor seja o certificado digital da aplicação parceira, emitido por uma autoridade certificadora (AC) reconhecida pelo serviço *web*.

- **Implantação em servidores virtualizados (na nuvem)**

Com o objetivo de experimentar um serviço de computação em nuvem (*cloud computing*), os sistemas que compõem esta prova de conceito devem ser implantados em servidores virtualizados na plataforma *Elastic Compute Cloud* (EC2) da Amazon. Para tanto, uma conta gratuita de experimentação por um ano foi criada, o que dá direito (sem custos adicionais) a setecentos e cinquenta horas mensais de microinstâncias T1 virtualizadas (AMAZON, 2012).

Uma microinstância T1 tem a seguinte configuração de *hardware*:

- Até 2 unidades de processamento EC2 (valor máximo de pico, em curtos intervalos de demanda);
- Memória (RAM) de 613 MB;
- Armazenamento somente através de *Elastic Block Store* (EBS);
- Plataforma de 32 ou 64 bits;
- Baixo desempenho de Entrada/Saída (I/O).

Essas restrições de *hardware* implicam na necessidade de otimizar cada um dos servidores virtualizados, tanto durante a instalação do sistema operacional quanto na fase de configuração dos servidores de aplicação. Maiores informações podem ser encontradas no apêndice B deste trabalho, “Implantação na Nuvem”.

- **Portabilidade das aplicações *web***

É desejável que os usuários das aplicações possam acessá-las a partir de qualquer um dos mais populares navegadores *web* e sistemas operacionais do mercado.

Para fins de demonstração, as aplicações desta prova de conceito devem ser compatíveis com os navegadores *web*: *Microsoft Internet Explorer*, *Mozilla Firefox* e *Google Chrome*; e com os sistemas operacionais: *Microsoft Windows* e *GNU/Linux*.

3.3.2 Riscos

Os riscos tecnológicos e arquiteturais no desenvolvimento da solução são os seguintes:

ID	Título	Descrição	Impacto	Complexidade	Ação de Mitigação
R1	Impossibilidade de emissão de certificados	Ausência de uma infraestrutura para emissão de certificados digitais.	Alto	Média	1. Instituição de uma ICP própria; 2. Criação de uma aplicação para administrar a ICP.
R2	Ausência de *smart cards*	Dificuldade na obtenção de *smart cards,* necessários para o armazenamento de certificados.	Alto	Baixa	1. Implementação de uma funcionalidade que permite ler certificados em arquivos PKCS#12.
R3	Dificuldade de implantação na nuvem	Devido às restrições da conta *one year free* da Amazon, a implantação dos servidores JEE pode se tornar inviável.	Baixo	Alta	1. Otimização das configurações dos servidores, incluindo parâmetros da JVM; 2. Avaliar custos de manter instâncias maiores na Amazon; 3. Em caso de impossibilidade total, abandonar essa abordagem, pois não é essencial a esta prova de conceito.
R4	Problemas com instalação de *drivers* de dispositivos em computadores do usuário	Pela necessidade da instalação de um leitor de *smart cards* e do próprio cartão em si, o usuário poderá não conseguir instalar os *drivers* requeridos, por motivos diversos.	Alto	Alta	1. Evitar necessidade de instalar *drivers* PKCS#11 no navegador *web*; 2. Isso implica na não-utilização de autenticação CLIENT-CERT na solução; 3. Implementação de uma funcionalidade que permite ler certificados em arquivos PKCS#12.

Quadro 7 - Riscos tecnológicos e arquiteturais

3.3.3 Visão de Casos de Uso

Nesta seção são apresentados os principais casos de uso dos sistemas que compõem a prova de conceito, e destacados os que envolvem funcionalidades significativas à integração entre esses sistemas.

- **Serviço de Identificação do Cidadão (SICid):**

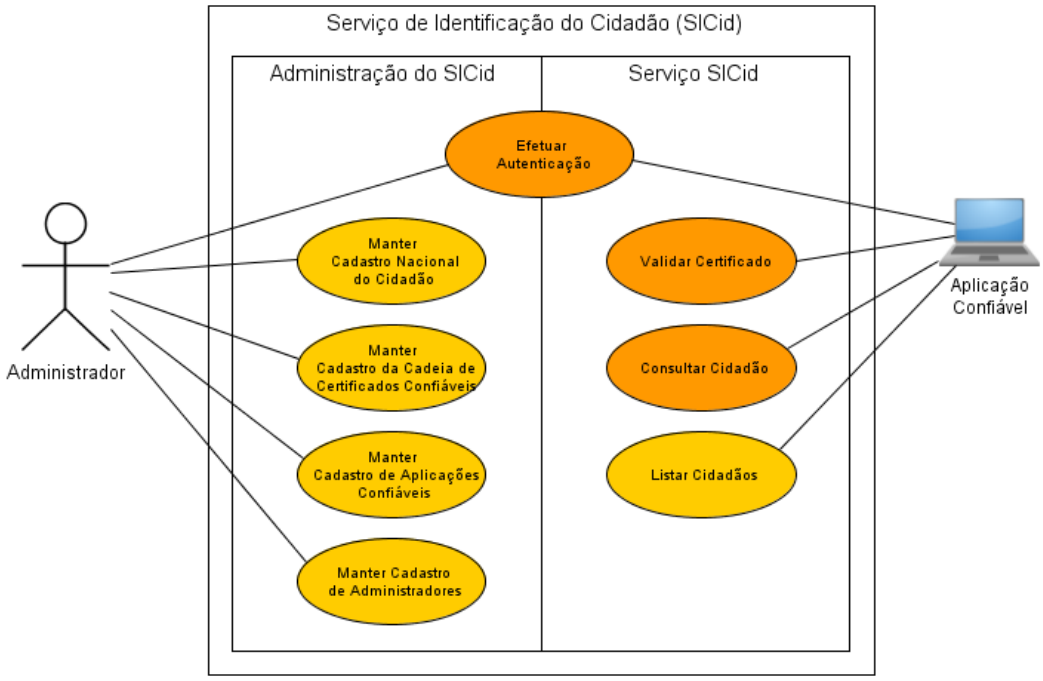

Figura 11 - Diagrama de casos de uso do serviço SICid

Serviço de Identificação do Cidadão (SICid) / Caso de uso: Efetuar Autenticação	
Caso de Uso:	Efetuar Autenticação.
Atores:	Administrador, Aplicação Confiável.
Finalidade:	Identificar o usuário da aplicação administrativa do serviço; ou Identificar a aplicação consumidora do serviço.
Visão Geral:	Um Administrador acessa a aplicação administrativa (SICid Admin), utilizando seu cartão de identificação. O sistema deve conceder acesso conforme as regras de autorização; ou Uma Aplicação acessa um serviço oferecido pelo SICid (para consumi-lo), usando seu certificado digital de aplicação. O serviço deve conceder acesso conforme as regras de autorização.
Pré-condições:	Nenhuma.
Pós-condições:	Administrador autorizado; ou Aplicação confiável autorizada.

Quadro 8 - Caso de uso: Efetuar Autenticação - SICid

Serviço de Identificação do Cidadão (SICid) / Caso de uso: Validar Certificado	
Caso de Uso:	Validar Certificado.
Atores:	Aplicação Confiável.
Finalidade:	Oferecer a validação de um certificado digital, através de um serviço.
Visão Geral:	Uma Aplicação acessa o serviço de validação, enviando o conteúdo de um certificado digital. O serviço retorna uma informação de *status* sobre o certificado, indicando: se ele é válido, foi revogado, está expirado ou não foi emitido por uma Autoridade Certificadora (AC) confiável (inválido).
Pré-condições:	Aplicação deve estar autorizada (Caso de Uso: Efetuar Autenticação).
Pós-condições:	Nenhuma.

Quadro 9 - Caso de uso: Validar Certificado - SICid

Serviço de Identificação do Cidadão (SICid) / Caso de uso: Consultar Cidadão	
Caso de Uso:	Consultar Cidadão.
Atores:	Aplicação Confiável.
Finalidade:	Oferecer dados cadastrais de um cidadão, através de um serviço.
Visão Geral:	Uma Aplicação acessa o serviço de consulta de cidadão, enviando o conteúdo do certificado digital pertencente ao cidadão a ser consultado. O serviço retorna os dados do cidadão constantes no Cadastro Nacional do Cidadão.
Pré-condições:	Aplicação deve estar autorizada (Caso de Uso: Efetuar Autenticação).
Pós-condições:	Nenhuma.

Quadro 10 - Caso de uso: Consultar Cidadão - SICid

- **Banco Seguro:**

Figura 12 - Diagrama de casos de uso do sistema Banco Seguro

Banco Seguro / Caso de uso: Efetuar Autenticação	
Caso de Uso:	Efetuar Autenticação.
Atores:	Cliente, Gerente, Aplicação Confiável.
Finalidade:	Identificar o usuário (Cliente ou Gerente) do banco; ou Identificar a aplicação consumidora do serviço bancário.
Visão Geral:	Um Cliente ou Gerente acessa a aplicação de *internet banking* do Banco Seguro, utilizando seu cartão de identificação. O sistema deve conceder acesso conforme as regras de autorização; ou Uma Aplicação acessa um serviço oferecido pelo Banco Seguro (para consumi-lo), usando seu certificado digital de aplicação. O serviço deve conceder acesso conforme as regras de autorização.
Pré-condições:	Nenhuma.
Pós-condições:	Cliente ou Gerente autorizado; ou Aplicação confiável autorizada.

Quadro 11 - Caso de uso: Efetuar Autenticação - Banco Seguro

Banco Seguro / Caso de uso: Pagar Títulos ou Documentos	
Caso de Uso:	Pagar Títulos ou Documentos.
Atores:	Cliente, Gerente, Aplicação Confiável.
Finalidade:	Oferecer a funcionalidade de pagamento de títulos (contas ou boletos) a clientes que possuam conta no banco, através do *internet banking* ou através de um serviço acessível por outras aplicações.
Visão Geral:	Um Cliente ou Gerente acessa a aplicação de *internet banking*, e realiza o pagamento informando seu valor. O sistema então, debita o valor correspondente a partir de sua conta e atualiza o extrato; ou Uma Aplicação acessa o serviço de pagamento de títulos, enviando o conteúdo do certificado digital do cliente do banco (Cliente ou Gerente), e o valor a ser pago. O serviço então, debita o valor correspondente a partir de sua conta, atualiza o extrato, e retorna um *status* da operação.
Pré-condições:	Cliente, Gerente ou Aplicação deve estar autorizado (Caso de Uso: Efetuar Autenticação). Cliente ou Gerente deve possuir conta aberta no banco.
Pós-condições:	Saldo do Cliente ou Gerente atualizado. Extrato atualizado.

Quadro 12 - Caso de uso: Pagar Títulos ou Documentos - Banco Seguro

- **Receita Nacional:**

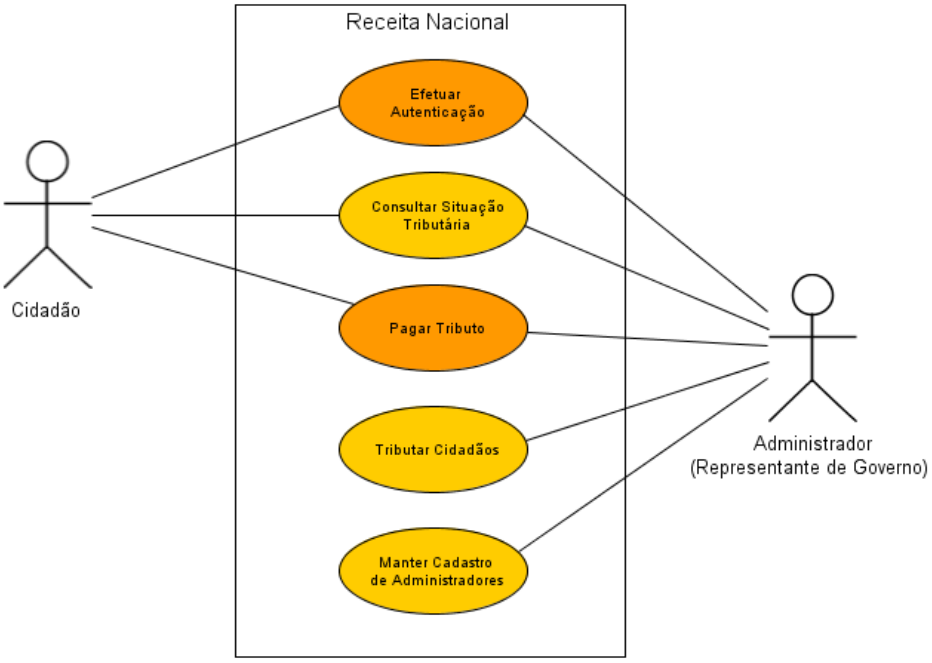

Figura 13 - Diagrama de casos de uso do sistema Receita Nacional

Receita Nacional / Caso de uso: Efetuar Autenticação	
Caso de Uso:	Efetuar Autenticação.
Atores:	Cidadão, Administrador (Representante de Governo).
Finalidade:	Identificar o usuário (Cidadão ou Administrador) da aplicação da Receita Nacional.
Visão Geral:	Um Cidadão ou Administrador (Representante de Governo) acessa a aplicação da Receita Nacional, utilizando seu cartão de identificação. O sistema deve conceder acesso conforme as regras de autorização.
Pré-condições:	Nenhuma.
Pós-condições:	Cidadão ou Administrador (Representante de Governo) autorizado.

Quadro 13 - Caso de uso: Efetuar Autenticação - Receita Nacional

Receita Nacional / Caso de uso: Pagar Tributo	
Caso de Uso:	Pagar Tributo.
Atores:	Cidadão, Administrador (Representante de Governo).
Finalidade:	Oferecer ao cidadão a funcionalidade de pagamento de um tributo (imposto) devido, com a possibilidade de usar, como forma de pagamento, débito direto (*online*) em uma conta no Banco Seguro.
Visão Geral:	Um Cidadão ou Administrador (Representante de Governo) acessa a aplicação da Receita Nacional, e realiza o pagamento de um imposto devido. O sistema então, oferece a opção de pagamento via débito direto em uma conta no Banco Seguro. O Banco Seguro é uma empresa conveniada (parceira) da Receita. Após confirmação do usuário, o sistema acessa o serviço do Banco Seguro para efetuar a transação. O sistema então, retorna o *status* dessa transação (se foi bem sucedida, se usuário não possui conta no banco, se saldo foi insuficiente, etc).
Pré-condições:	Cidadão ou Administrador (Representante de Governo) deve estar autorizado (Caso de Uso: Efetuar Autenticação). Receita Nacional deve ser uma aplicação confiável para o serviço do Banco Seguro.
Pós-condições:	Tributo pago (situação regular, sem pendências).

Quadro 14 - Caso de uso: Pagar Tributo - Receita Nacional

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

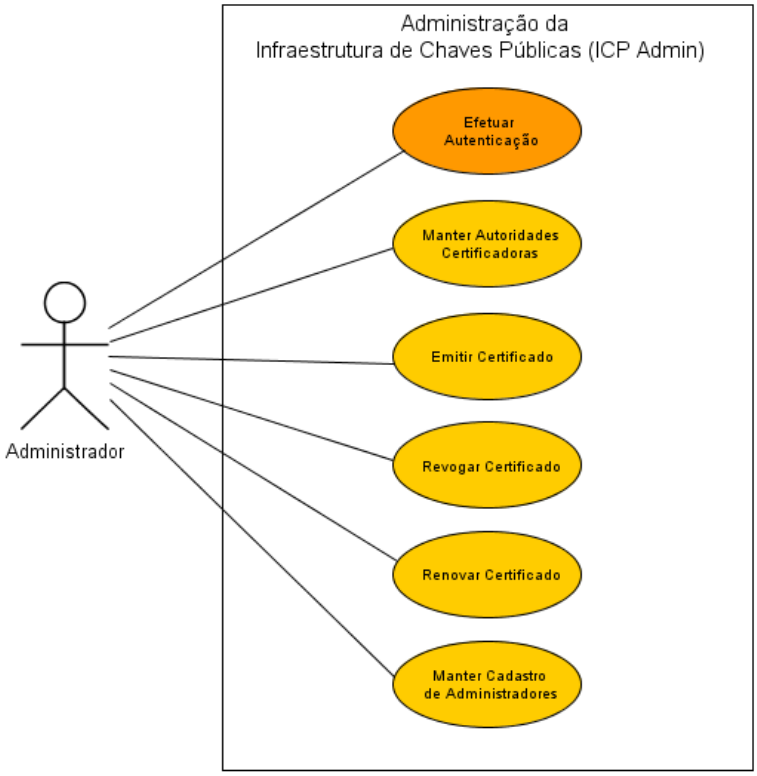

Figura 14 - Diagrama de casos de uso do sistema ICP Admin

Administração da Infraestrutura de Chaves Públicas (ICP Admin) Caso de uso: Efetuar Autenticação	
Caso de Uso:	Efetuar Autenticação.
Atores:	Administrador.
Finalidade:	Identificar o usuário da aplicação administrativa da ICP.
Visão Geral:	Um Administrador acessa a aplicação administrativa da ICP, utilizando seu cartão de identificação. O sistema deve conceder acesso conforme as regras de autorização.
Pré-condições:	Nenhuma.
Pós-condições:	Administrador autorizado.

Quadro 15 - Caso de uso: Efetuar Autenticação - ICP Admin

3.3.3.1 Realização de Casos de Uso

Os casos de uso que envolvem funcionalidades significativas à integração de sistemas são detalhados nesta seção, através de sua expansão, em forma real (LARMAN, 2008).

A descrição do fluxo principal de cada caso de uso segue uma sequência de eventos, sugerindo a interação entre os sistemas e atores envolvidos.

- **Serviço de Identificação do Cidadão (SICid):**

Caso de uso: Efetuar Autenticação

Ator: Administrador

Fluxo Principal

Ação do Ator	**Resposta do Sistema**
1. Este caso de uso inicia quando um Administrador acessa a aplicação administrativa do SICid (SICid Admin).	
	2. Exibe a tela de autenticação (*login*), contendo a *applet* responsável por ler o certificado digital armazenado no cartão de identificação (*smart card*).
3. Administrador insere o cartão na leitora (se já não o tiver realizado antes) e confirma.	
	4. Exibe a lista de dispositivos criptográficos (*smart cards* e *tokens*) presentes no computador do usuário Administrador.
5. Seleciona o dispositivo correspondente ao cartão de identificação e confirma.	
	6. Exibe a solicitação da senha (PIN) que protege o dispositivo.
7. Informa a senha (PIN) do cartão de identificação.	
	8. Exibe a lista de certificados armazenados dentro do dispositivo criptográfico (cartão).
9. Seleciona o certificado que o identifica e confirma.	

Ação do Ator	Resposta do Sistema
	10. Assina (digitalmente) o conteúdo do certificado digital, usando a chave privada correspondente, presente no dispositivo criptográfico.
	11. A *applet* envia o conteúdo do certificado e sua assinatura (ambos codificados em Base64) ao servlet da aplicação (por meio de HTTP POST).
	12. A aplicação (SICid Admin) realiza autenticação no serviço SICid (*web service*), usando seu certificado de aplicação.
	13. O serviço SICid procede a autenticação, conforme fluxo do caso de uso “Efetuar Autenticação, Ator: Aplicação Confiável”.
	14. A aplicação (SICid Admin) invoca o serviço “Validar Certificado” do SICid, conforme fluxo do caso de uso “Validar Certificado”, passando para o serviço o certificado do usuário Administrador.
	15. Após receber o *status* de VALIDO para o certificado do Administrador, a aplicação (SICid Admin) busca as credenciais desse usuário em seu banco de dados (entidade Usuario), obtendo seu perfil (role) de Administrador.
	16. Autoriza o acesso ao Administrador.
17. Administrador autorizado pode realizar as operações dentro da aplicação.	

Caso de uso: Efetuar Autenticação

Ator: Aplicação Confiável

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando uma Aplicação Confiável deseja acessar (consumir) um serviço do SICid.	
	2. Exige autenticação BASIC, contendo nos campos *username* e *password*, os dados: *username* = conteúdo do certificado digital da aplicação, codificado em Base64. *password* = assinatura digital (RSA/SHA1) do conteúdo do certificado, codificado em Base64.

Ação do Ator	Resposta do Sistema
3. Envia ao SICid o conteúdo de seu certificado de aplicação e a assinatura correspondente, através da autenticação BASIC.	
	4. Verifica a assinatura recebida (se o conteúdo do certificado foi mesmo assinado com a chave privada correspondente - de propriedade exclusiva da aplicação consumidora, dona do certificado).
	5. Busca as credenciais da aplicação em seu banco de dados (entidade ConsumidorConfiavel), obtendo o perfil (role) de Aplicação Confiável.
	6. Autoriza o acesso à Aplicação Confiável.
7. Aplicação Confiável autorizada pode consumir os serviços oferecidos.	

Caso de uso: Validar Certificado

Ator: Aplicação Confiável

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando uma Aplicação Confiável consome o serviço “Validar Certificado”.	
2. Pré-condição: Efetuar Autenticação (Aplicação Confiável).	
3. Envia ao SICid o conteúdo do certificado a ser validado, codificado em Base64.	
	4. Verifica integridade do conteúdo do certificado.
	5. Verifica se certificado é autoassinado.
	6. Verifica se certificado tem sua data de validade expirada.
	7. Verifica se certificado foi emitido por uma Autoridade Certificadora (AC) confiável, presente na cadeia cadastrada em seu banco de dados (entidade CertificadoConfiavel).

Ação do Ator	Resposta do Sistema
	8. Verifica se certificado foi revogado, contactando as Listas de Certificados Revogados (LCR), cujos *links* são descritos no próprio certificado.
	9. Retorna um *status*, conforme o resultado da validação do certificado.
10. Obtém o *status* de validação, a partir da resposta do serviço SICid.	

Caso de uso: Consultar Cidadão

Ator: Aplicação Confiável

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando uma Aplicação Confiável consome o serviço “Consultar Cidadão”.	
2. Pré-condição: Efetuar Autenticação (Aplicação Confiável).	
3. Envia ao SICid o conteúdo do certificado do cidadão a ser consultado, codificado em Base64.	
	4. Verifica integridade do conteúdo do certificado.
	5. Busca no banco de dados do Cadastro Nacional do Cidadão (entidade Cidadao) os dados cadastrais. Extrai, a partir do certificado, informações adicionais (e temporárias, não persistidas), como e-mail e nome de *login*.
	6. Retorna os dados cadastrais do cidadão e as informações adicionais presentes no seu certificado.
7. Obtém os dados cadastrais de um cidadão, a partir da resposta do serviço SICid.	

- **Banco Seguro:**

Caso de uso: Efetuar Autenticação

Atores: Gerente, Cliente

Fluxo Principal

Ação do Ator	**Resposta do Sistema**
1. Este caso de uso inicia quando um Gerente ou Cliente acessa a aplicação *web* do Banco Seguro (*internet banking*).	
	2. Exibe a tela de autenticação (*login*), contendo a *applet* responsável por ler o certificado digital armazenado no cartão de identificação (*smart card*). Esta *applet* faz parte do serviço SICid.
3. Administrador insere o cartão na leitora (se já não o tiver realizado antes) e confirma.	
	4. Exibe a lista de dispositivos criptográficos (*smart cards* e *tokens*) presentes no computador do usuário Administrador.
5. Seleciona o dispositivo correspondente ao cartão de identificação e confirma.	
	6. Exibe a solicitação da senha (PIN) que protege o dispositivo.
7. Informa a senha (PIN) do cartão de identificação.	
	8. Exibe a lista de certificados armazenados dentro do dispositivo criptográfico (cartão).
9. Seleciona o certificado que o identifica e confirma.	
	10. Assina (digitalmente) o conteúdo do certificado digital, usando a chave privada correspondente, presente no dispositivo criptográfico.
	11. A *applet* envia o conteúdo do certificado e sua assinatura (ambos codificados em Base64) ao servlet da aplicação (por meio de HTTP POST).
	12. A aplicação (Banco Seguro) realiza autenticação no serviço SICid (*web service*), usando seu certificado de aplicação.

Ação do Ator	Resposta do Sistema
	13. O serviço SICid procede a autenticação, conforme fluxo do caso de uso “Efetuar Autenticação, Ator: Aplicação Confiável”.
	14. A aplicação (Banco Seguro) invoca o serviço “Validar Certificado” do SICid, conforme fluxo do caso de uso “Validar Certificado”, passando para o serviço o certificado do usuário Gerente ou Cliente.
	15. Após receber o *status* de VALIDO para o certificado do Gerente ou Cliente, a aplicação (Banco Seguro) busca as credenciais desse usuário em seu banco de dados (entidade Usuario), obtendo seu perfil (role) de Gerente ou Cliente.
	16. Autoriza o acesso ao Gerente ou Cliente.
17. Gerente ou Cliente autorizado pode realizar as operações (as quais tem direito) dentro da aplicação.	

Caso de uso: Efetuar Autenticação

Ator: Aplicação Confiável

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando uma Aplicação Confiável deseja acessar (consumir) um serviço do Banco Seguro.	
	2. Exige autenticação BASIC, contendo nos campos *username* e *password*, os dados: *username* = conteúdo do certificado digital da aplicação, codificado em Base64. *password* = assinatura digital (RSA/SHA1) do conteúdo do certificado, codificado em Base64.
3. Envia ao Banco Seguro o conteúdo de seu certificado de aplicação e a assinatura correspondente, através da autenticação BASIC.	
	4. Verifica a assinatura recebida (se o conteúdo do certificado foi mesmo assinado com a chave privada correspondente - de propriedade exclusiva da aplicação consumidora, dona do certificado).

Ação do Ator	Resposta do Sistema
	5. Busca as credenciais da aplicação em seu banco de dados (entidade ConsumidorConfiavel), obtendo o perfil (role) de Aplicação Confiável.
	6. Autoriza o acesso à Aplicação Confiável.
7. Aplicação Confiável autorizada pode consumir os serviços oferecidos.	

Caso de uso: Pagar Títulos ou Documentos

Ator: Aplicação Confiável

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando uma Aplicação Confiável consome o serviço “Pagar Títulos ou Documentos”.	
2. Pré-condição: Efetuar Autenticação (Aplicação Confiável).	
3. Envia ao serviço do Banco Seguro o conteúdo do certificado digital do cliente do banco, codificado em Base64, e o valor do documento a ser pago.	
	4. O serviço do Banco Seguro verifica integridade do conteúdo do certificado.
	5. O serviço do Banco Seguro realiza autenticação no serviço SICid (*web service*), usando seu certificado de aplicação.
	6. O serviço SICid procede a autenticação, conforme fluxo do caso de uso do SICid, “Efetuar Autenticação, Ator: Aplicação Confiável”.
	7. O serviço do Banco Seguro invoca o serviço “Validar Certificado” do SICid, conforme fluxo do caso de uso do SICid, “Validar Certificado”, passando o certificado do cliente do banco.
	8. Após receber o *status* de VALIDO para o certificado do cliente do banco, o serviço do Banco Seguro invoca o serviço “Consultar Cidadão” do SICid, conforme fluxo do caso de uso do SICid, “Consultar Cidadão”.

Ação do Ator	Resposta do Sistema
	9. Após receber os dados cadastrais do cidadão, o serviço do Banco Seguro busca em sua base de dados as informações de sua conta – ou seja, um cidadão que é cliente do banco (entidade Conta).
	10. O serviço do Banco Seguro realiza a movimentação da conta (no caso, a operação de pagamento, realizando o débito do valor especificado) e atualiza o extrato da conta (entidade Extrato).
	11. Retorna um *status*, conforme o resultado da operação bancária.
12. Obtém o *status* da transação bancária, a partir da resposta do serviço do Banco Seguro.	

- **Receita Nacional:**

Caso de uso: Efetuar Autenticação

Atores: Cidadão, Administrador (Representante de Governo)

Fluxo Principal

Ação do Ator	Resposta do Sistema
1. Este caso de uso inicia quando um Cidadão ou Administrador (Representante de Governo) acessa a aplicação *web* da Receita Nacional.	
	2. Exibe a tela de autenticação (*login*), contendo a *applet* responsável por ler o certificado digital armazenado no cartão de identificação (*smart card*). Esta *applet* faz parte do serviço SICid.
3. Administrador insere o cartão na leitora (se já não o tiver realizado antes) e confirma.	
	4. Exibe a lista de dispositivos criptográficos (*smart cards* e *tokens*) presentes no computador do usuário Administrador.
5. Seleciona o dispositivo correspondente ao cartão de identificação e confirma.	
	6. Exibe a solicitação da senha (PIN) que protege o dispositivo.
7. Informa a senha (PIN) do cartão de identificação.	

Ação do Ator	Resposta do Sistema
	8. Exibe a lista de certificados armazenados dentro do dispositivo criptográfico (cartão).
9. Seleciona o certificado que o identifica e confirma.	
	10. Assina (digitalmente) o conteúdo do certificado digital, usando a chave privada correspondente, presente no dispositivo criptográfico.
	11. A *applet* envia o conteúdo do certificado e sua assinatura (ambos codificados em Base64) ao servlet da aplicação (por meio de HTTP POST).
	12. A aplicação (Receita Nacional) realiza autenticação no serviço SICid (*web service*), usando seu certificado de aplicação.
	13. O serviço SICid procede a autenticação, conforme fluxo do caso de uso “Efetuar Autenticação, Ator: Aplicação Confiável”.
	14. A aplicação (Receita Nacional) invoca o serviço “Validar Certificado” do SICid, conforme fluxo do caso de uso “Validar Certificado”, passando para o serviço o certificado do usuário Cidadão ou Administrador.
	15. Após receber o *status* de VALIDO para o certificado do Cidadão ou Administrador, a aplicação (Receita Nacional) busca as credenciais desse usuário em seu banco de dados (entidade Cidadao), obtendo seu perfil (role) de Cidadão ou Administrador (Representante de Governo).
	16. Autoriza o acesso ao Cidadão ou Administrador.
17. Cidadão ou Administrador autorizado pode realizar as operações (as quais tem direito) dentro da aplicação.	

Caso de uso: Pagar Tributo

Atores: Cidadão, Administrador (Representante de Governo)

Fluxo Principal

Ação do Ator	**Resposta do Sistema**
1. Este caso de uso inicia quando um Cidadão ou Administrador (Representante de Governo) acessa a funcionalidade “Pagar Tributo” da aplicação *web* da Receita Nacional.	
2. Pré-condição: Efetuar Autenticação.	
	3. Exibe as opções de formas de pagamento, e dentre elas, a de pagar através de débito *online* com o Banco Seguro (que é uma empresa conveniada com a Receita Nacional).
4. Seleciona a opção de pagar o tributo através de uma conta no Banco Seguro e confirma.	
	5. A aplicação (Receita Nacional) realiza autenticação no serviço do Banco Seguro (*web service*), usando seu certificado de aplicação.
	6. O serviço do Banco Seguro procede a autenticação, conforme fluxo do caso de uso do Banco Seguro, “Efetuar Autenticação, Ator: Aplicação Confiável”.
	7. A aplicação (Receita Nacional) invoca o serviço “Pagar Títulos ou Documentos” do Banco Seguro, conforme fluxo do caso de uso “Pagar Títulos ou Documentos”, passando para o serviço o certificado do usuário Cidadão ou Administrador, e o valor do tributo a ser pago.
	8. Após receber do serviço do Banco Seguro o *status* de OPERAÇÃO REALIZADA COM SUCESSO, a aplicação (Receita Nacional) atualiza seu banco de dados (entidade Tributo), sinalizando que o tributo foi pago e a situação tributária está REGULAR.
	9. Exibe a situação tributária REGULAR para o usuário Cidadão ou Administrador (imposto devido foi pago).
10. Pós-condição: Situação tributária do Cidadão ou Administrador está REGULAR.	

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

Caso de uso: Efetuar Autenticação

Ator: Administrador

Fluxo Principal

Ação do Ator	**Resposta do Sistema**
1. Este caso de uso inicia quando um Administrador acessa a aplicação administrativa da ICP (ICP Admin).	
	2. Exibe a tela de autenticação (*login*), contendo a *applet* responsável por ler o certificado digital armazenado no cartão de identificação (*smart card*). Esta *applet* faz parte do serviço SICid.
3. Administrador insere o cartão na leitora (se já não o tiver realizado antes) e confirma.	
	4. Exibe a lista de dispositivos criptográficos (*smart cards* e *tokens*) presentes no computador do usuário Administrador.
5. Seleciona o dispositivo correspondente ao cartão de identificação e confirma.	
	6. Exibe a solicitação da senha (PIN) que protege o dispositivo.
7. Informa a senha (PIN) do cartão de identificação.	
	8. Exibe a lista de certificados armazenados dentro do dispositivo criptográfico (cartão).
9. Seleciona o certificado que o identifica e confirma.	
	10. Assina (digitalmente) o conteúdo do certificado digital, usando a chave privada correspondente, presente no dispositivo criptográfico.
	11. A *applet* envia o conteúdo do certificado e sua assinatura (ambos codificados em Base64) ao servlet da aplicação (por meio de HTTP POST).
	12. A aplicação (ICP Admin) realiza autenticação no serviço SICid (*web service*), usando seu certificado de aplicação.

Ação do Ator	Resposta do Sistema
	13. O serviço SICid procede a autenticação, conforme fluxo do caso de uso “Efetuar Autenticação, Ator: Aplicação Confiável”.
	14. A aplicação (ICP Admin) invoca o serviço “Validar Certificado” do SICid, conforme fluxo do caso de uso “Validar Certificado”, passando para o serviço o certificado do usuário Administrador.
	15. Após receber o *status* de VALIDO para o certificado do Administrador, a aplicação (ICPAdmin) busca as credenciais desse usuário em seu banco de dados (entidade Usuario), obtendo seu perfil (role) de Administrador.
	16. Autoriza o acesso ao Administrador.
17. Administrador autorizado pode realizar as operações dentro da aplicação.	

3.3.4 Visão de Aplicação

Aqui estão definidas as regras de nomeação de pacotes, domínios e projetos, relativos às aplicações que compõem a prova de conceito que demonstra este trabalho.

Prova de Conceito (PoC) - Integração de Sistemas: Simplificando a Vida do Cidadão		
Domínios:	`tcc.fiap.robsonmartins.com`	Domínio Principal (raiz)
	`icp.robsonmartins.com`	Arquivos da ICP Robson Martins.
Projetos-Fonte:	TccFiapCommon	Implementação comum a todas as aplicações da prova de conceito.
	TccFiapSite	Implementação do Portal de Entrada da prova de conceito (*site web* estático).
	TccFiapEApp	Implementação da Aplicação *Enterprise* (EAR), que inclui todas as aplicações da prova de conceito em um só pacote de implantação.
Pacotes:	`com.robsonmartins.fiap.tcc.util`	Classes úteis à implementação.
	`com.robsonmartins.fiap.tcc.dao`	Classes de acesso a dados (DAO), persistidos em banco de dados (ou em outros meios).

Quadro 16 - Nomeação de domínios, pacotes e projetos da prova de conceito

Nome da Aplicação: Serviço de Identificação do Cidadão (SICid)		
Domínios:	`/sicid`	Aplicação administrativa (SICid Admin).
	`/sicid/service`	Serviço *web*.
Projetos-Fonte:	SICidApplet	Implementação da *applet* de autenticação, responsável por ler certificados digitais armazenados em *smart cards* ou *tokens*.
	SICidCommon	Implementação comum aos módulos do SICid.
	SICidLoginModule	Implementação de *LoginModule* baseado em JAAS, que realiza a autenticação de um usuário a partir do serviço SICid.
	SICidEJB	Implementação da camada *model* do SICid (regras de negócio), usando EJB local.
	SICidService	Implementação do serviço (*web service*) do SICid, através de um *stateless* EJB.
	SICidWeb	Implementação da aplicação *web* que permite a administração do SICid (SICid Admin).
	SICidEApp	Implementação da Aplicação *Enterprise* (EAR), com o todos os módulos do SICid em um só pacote de implantação.
Pacotes:	`sicid.util`	Classes úteis à implementação.
	`sicid.bean`	Entidades (*entity beans*) e outros POJO's.
	`sicid.dao`	Classes de acesso a dados (DAO), persistidos em banco de dados (ou em outros meios).
	`sicid.model`	Classes da camada *model* (regras de negócio).
	`sicid.web.controller`	Classes da camada *control* (*managed beans*, e outras relacionadas com aplicações *web*).
	`sicid.ws`	Classes relacionadas a *web services* (serviços ou consumidores).

Quadro 17 - Nomeação de domínios, pacotes e projetos - SICid

Nome da Aplicação: Banco Seguro		
Domínios:	`/bancoseguro`	Aplicação *web* (*internet banking*).
	`/bancoseguro/service`	Serviço *web*.
Projetos-Fonte:	BancoSeguroCommon	Implementação comum aos módulos do Banco Seguro.
	BancoSeguroEJB	Implementação da camada *model* do Banco Seguro (regras de negócio), usando EJB local.
	BancoSeguroService	Implementação do serviço (*web service*) do Banco Seguro, através de um *stateless* EJB.
	BancoSeguroWeb	Implementação da aplicação *web* do Banco Seguro.
	BancoSeguroEApp	Implementação da Aplicação *Enterprise* (EAR), com o todos os módulos do Banco Seguro em um só pacote de implantação.
Pacotes:	`banco.bean`	Entidades (*entity beans*) e outros POJO's.
	`banco.dao`	Classes de acesso a dados (DAO), persistidos em banco de dados (ou em outros meios).
	`banco.model`	Classes da camada *model* (regras de negócio).
	`banco.web.controller`	Classes da camada *control* (*managed beans*, e outras relacionadas com aplicações *web*).
	`banco.ws`	Classes relacionadas a *web services* (serviços ou consumidores).

Quadro 18 - Nomeação de domínios, pacotes e projetos - Banco Seguro

Nome da Aplicação: Receita Nacional		
Domínios:	`/receita`	Aplicação *web*.
Projetos-Fonte:	ReceitaNacionalCommon	Implementação comum aos módulos da Receita Nacional.
	ReceitaNacionalEJB	Implementação da camada *model* da Receita Nacional (regras de negócio), usando EJB local.
	ReceitaNacional	Implementação da aplicação *web* da Receita Nacional.
	ReceitaNacionalEApp	Implementação da Aplicação *Enterprise* (EAR), com o todos os módulos da Receita Nacional em um só pacote de implantação.
Pacotes:	`receita.bean`	Entidades (*entity beans*) e outros POJO's.
	`receita.dao`	Classes de acesso a dados (DAO), persistidos em banco de dados (ou em outros meios).
	`receita.model`	Classes da camada *model* (regras de negócio).
	`receita.web.controller`	Classes da camada *control* (*managed beans*, e outras relacionadas com aplicações *web*).

Quadro 19 - Nomeação de domínios, pacotes e projetos - Receita Nacional

Nome da Aplicação: Administração da Infraestrutura de Chaves Públicas (ICP Admin)		
Domínios:	`/icpadmin`	Aplicação *web*.
Projetos-Fonte:	ICPAdminCommon	Implementação comum aos módulos do ICP Admin.
	ICPAdminEJB	Implementação da camada *model* do ICP Admin (regras de negócio), usando EJB local.
	ICPAdmin	Implementação da aplicação *web* do ICP Admin.
	ICPAdminEApp	Implementação da Aplicação *Enterprise* (EAR), com o todos os módulos do ICP Admin em um só pacote de implantação.
Pacotes:	`icp.util`	Classes úteis à implementação.
	`icp.bean`	Entidades (*entity beans*) e outros POJO's.
	`icp.dao`	Classes de acesso a dados (DAO), persistidos em banco de dados (ou em outros meios).
	`icp.model`	Classes da camada *model* (regras de negócio).
	`icp.web.controller`	Classes da camada *control* (*managed beans*, e outras relacionadas com aplicações *web*).

Quadro 20 - Nomeação de domínios, pacotes e projetos - ICP Admin

3.3.5 Visão Lógica

Esta seção descreve as partes significativas do ponto de vista da arquitetura do modelo de design, como sua organização em pacotes e subsistemas de serviço, e a organização desses subsistemas em camadas.

- **Serviço de Identificação do Cidadão (SICid):**

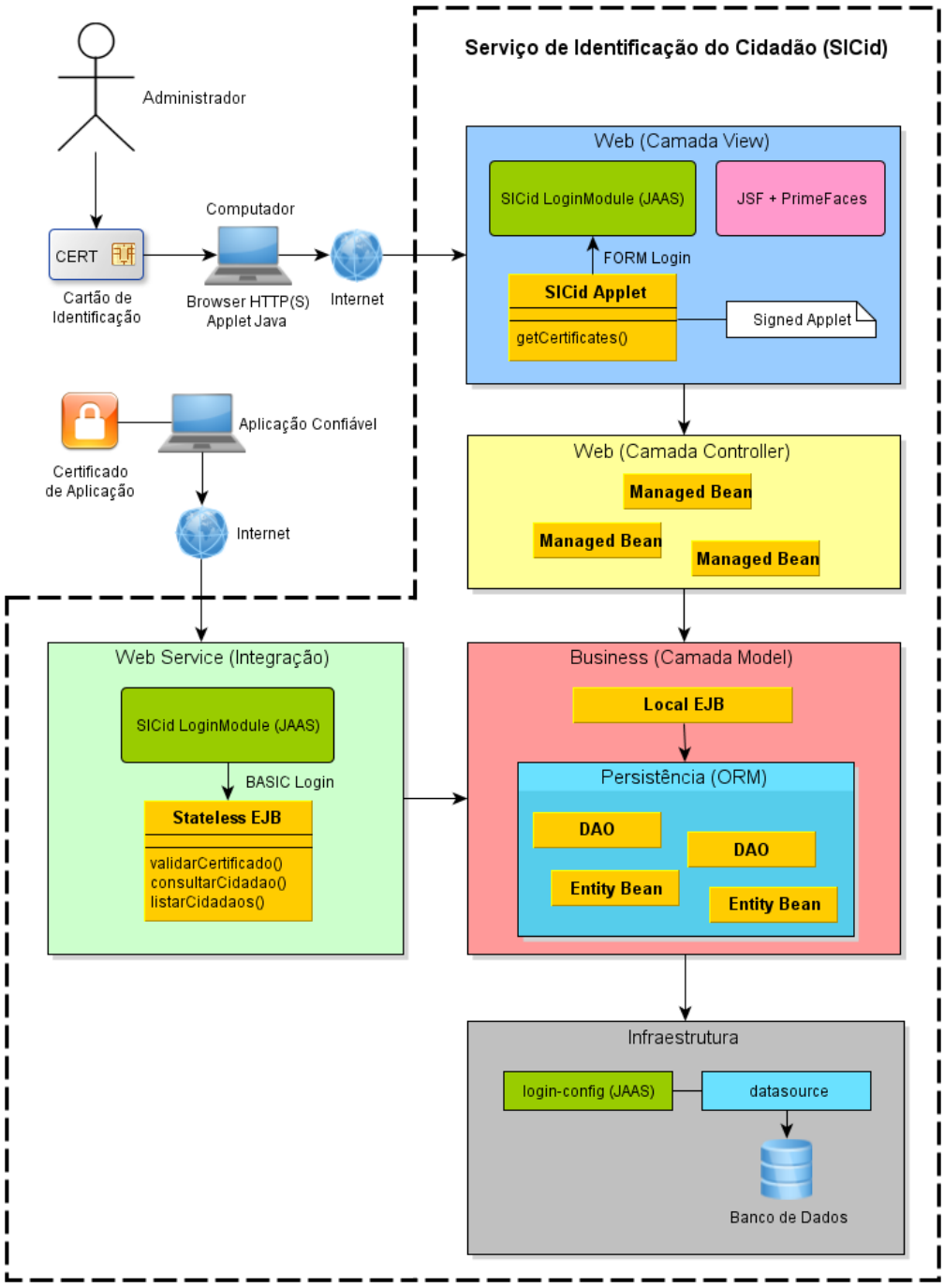

Figura 15 - Visão lógica do Serviço de Identificação do Cidadão (SICid)

- **Banco Seguro:**

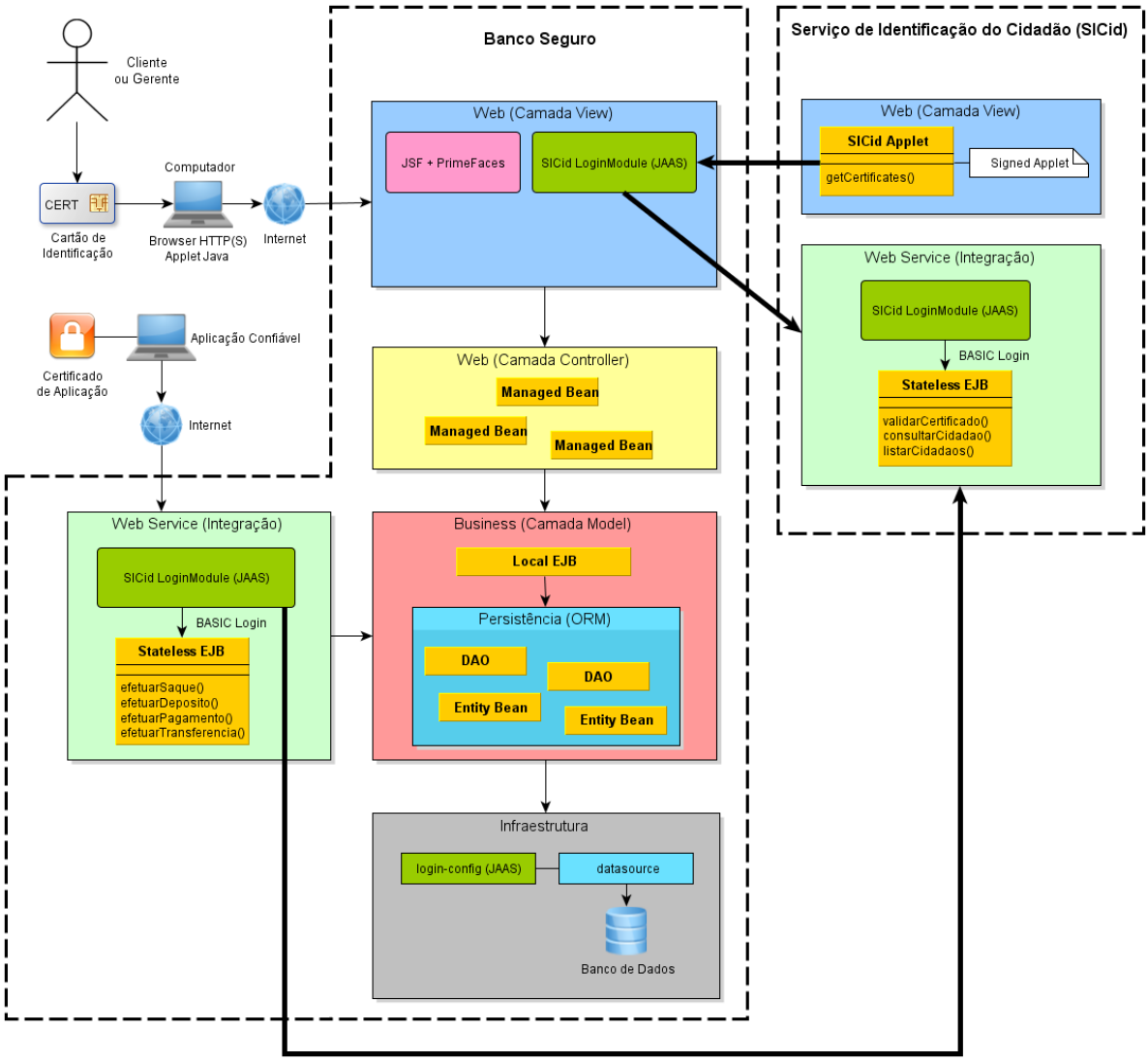

Figura 16 - Visão lógica do sistema Banco Seguro

- **Receita Nacional:**

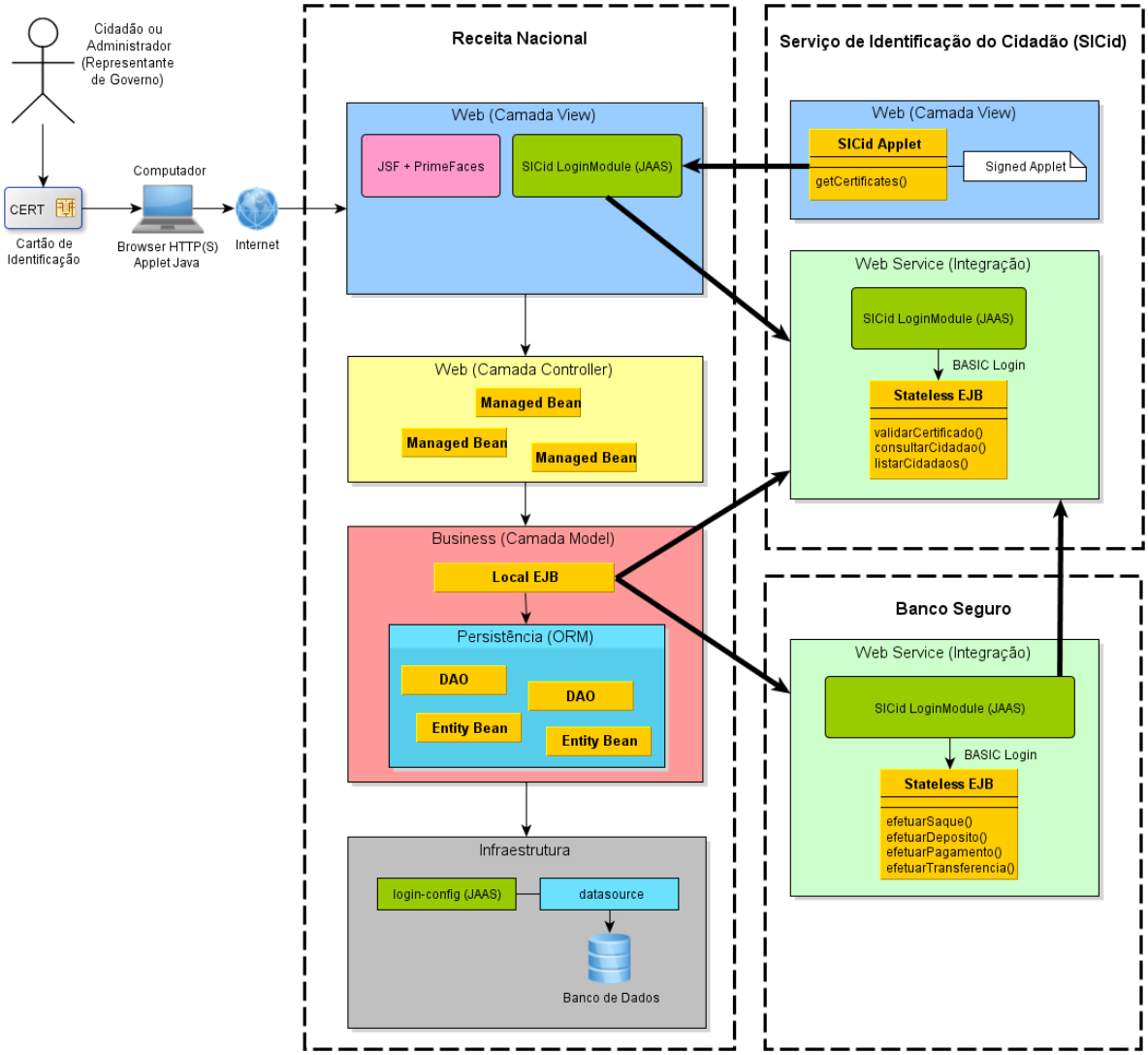

Figura 17 - Visão lógica do sistema Receita Nacional

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

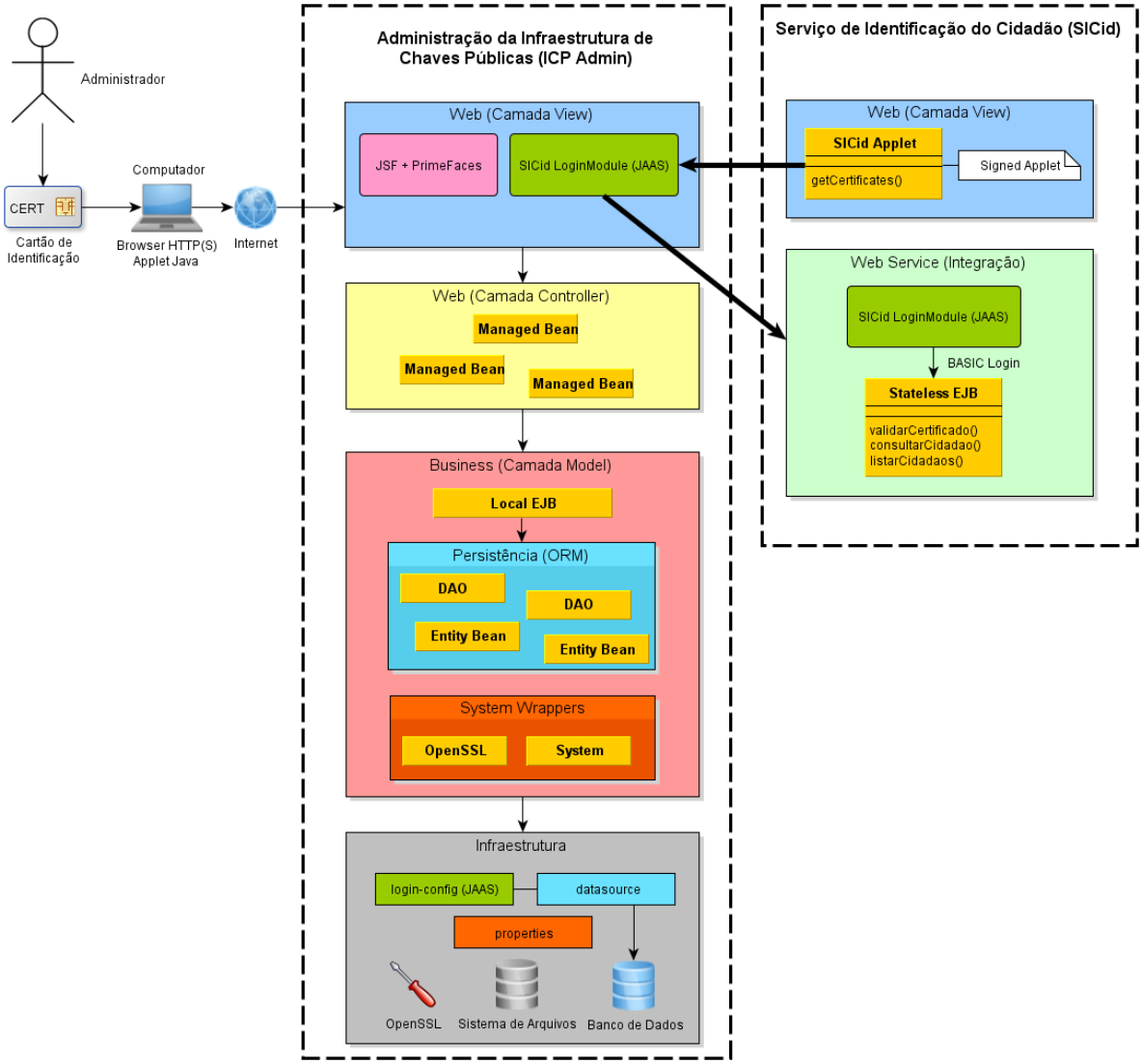

Figura 18 - Visão lógica do sistema ICP Admin

3.3.6 Visão de Implementação

Esta seção descreve a estrutura geral do modelo de implementação, a divisão do *software* em camadas e os subsistemas no modelo de implementação, além dos componentes significativos do ponto de vista da arquitetura.

- **Serviço de Identificação do Cidadão (SICid):**

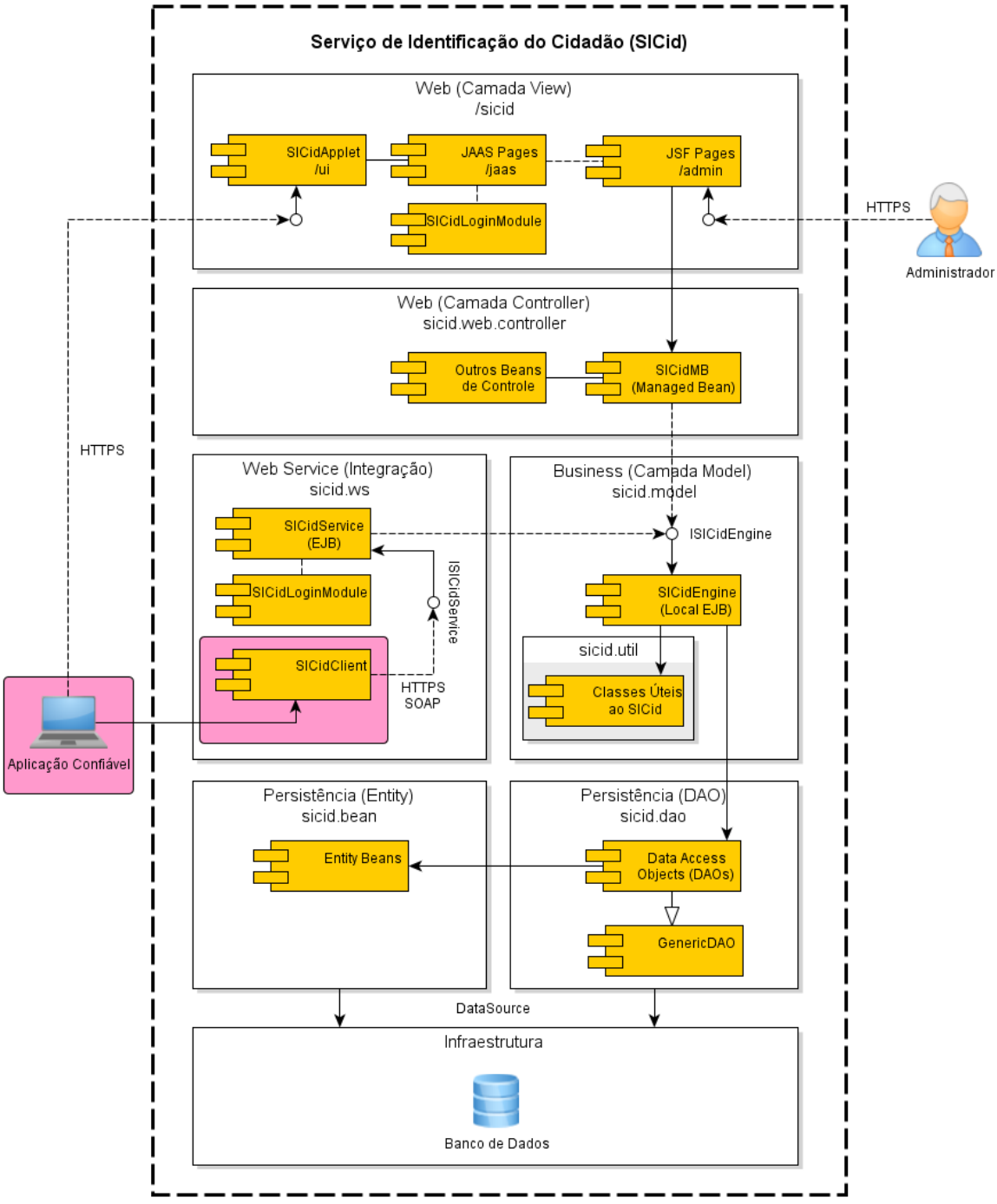

Figura 19 - Visão de implementação do serviço SICid

- **Banco Seguro:**

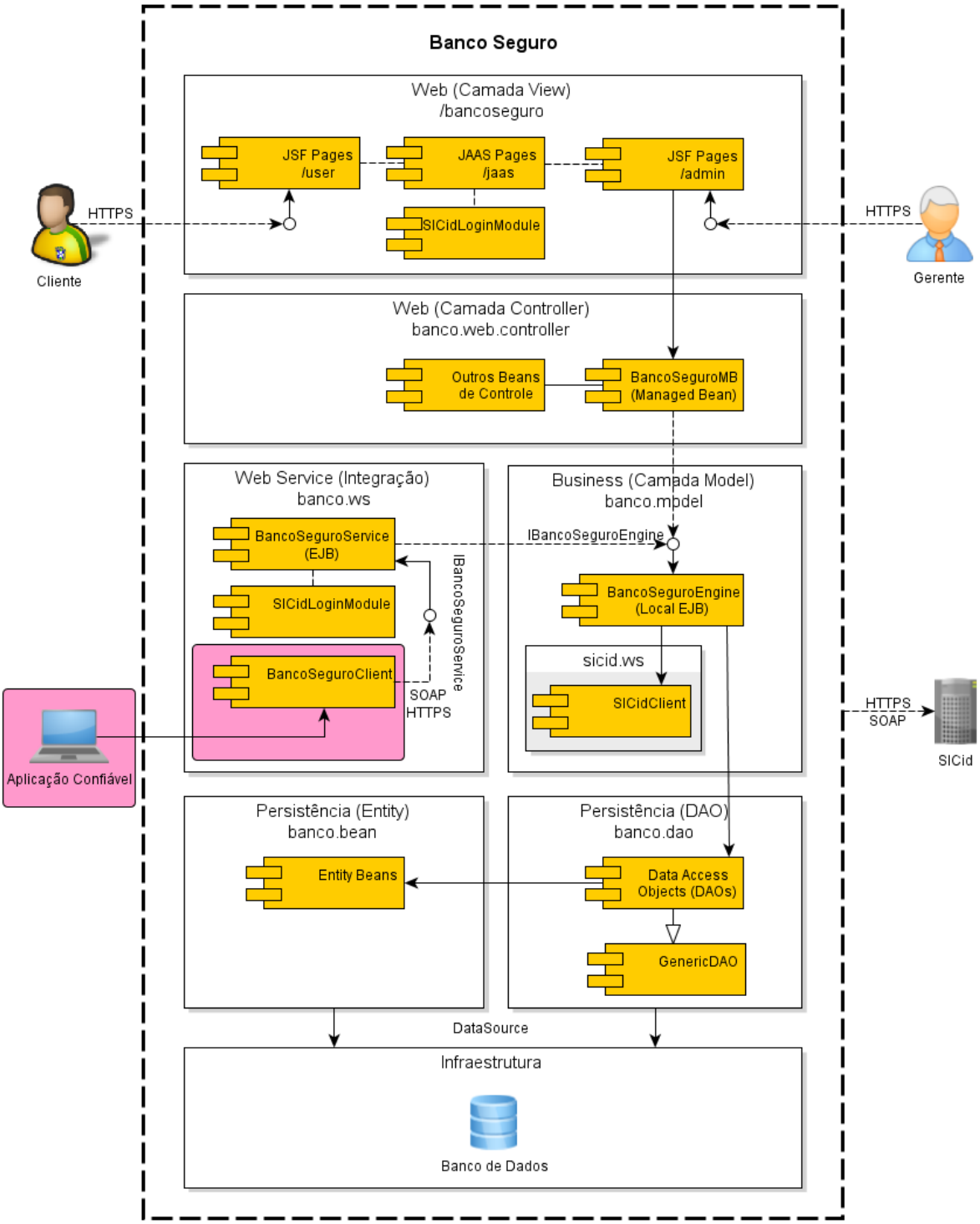

Figura 20 - Visão de implementação do sistema Banco Seguro

- **Receita Nacional:**

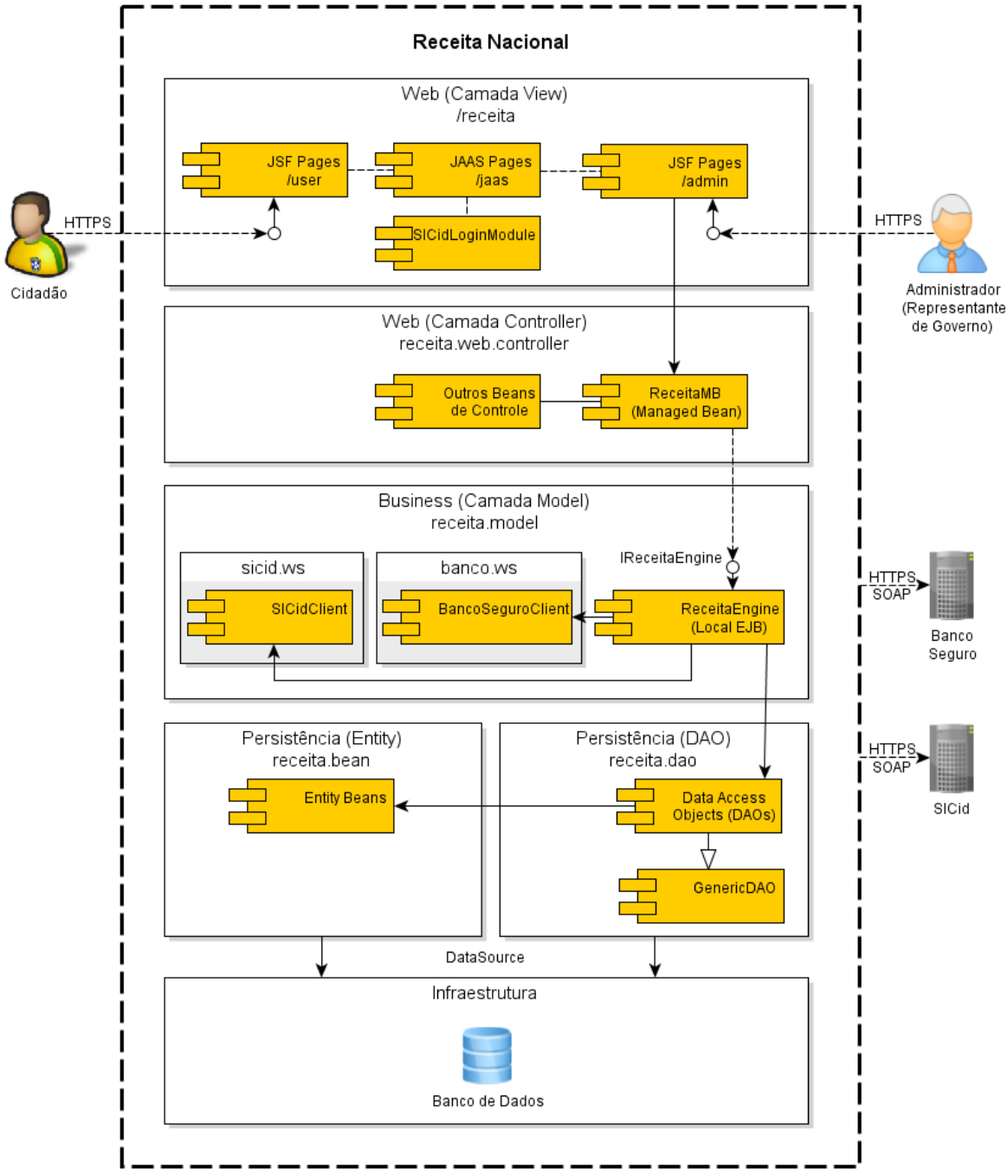

Figura 21 - Visão de implementação do sistema Receita Nacional

- **Administração da Infraestrutura de Chaves Públicas (ICP Admin):**

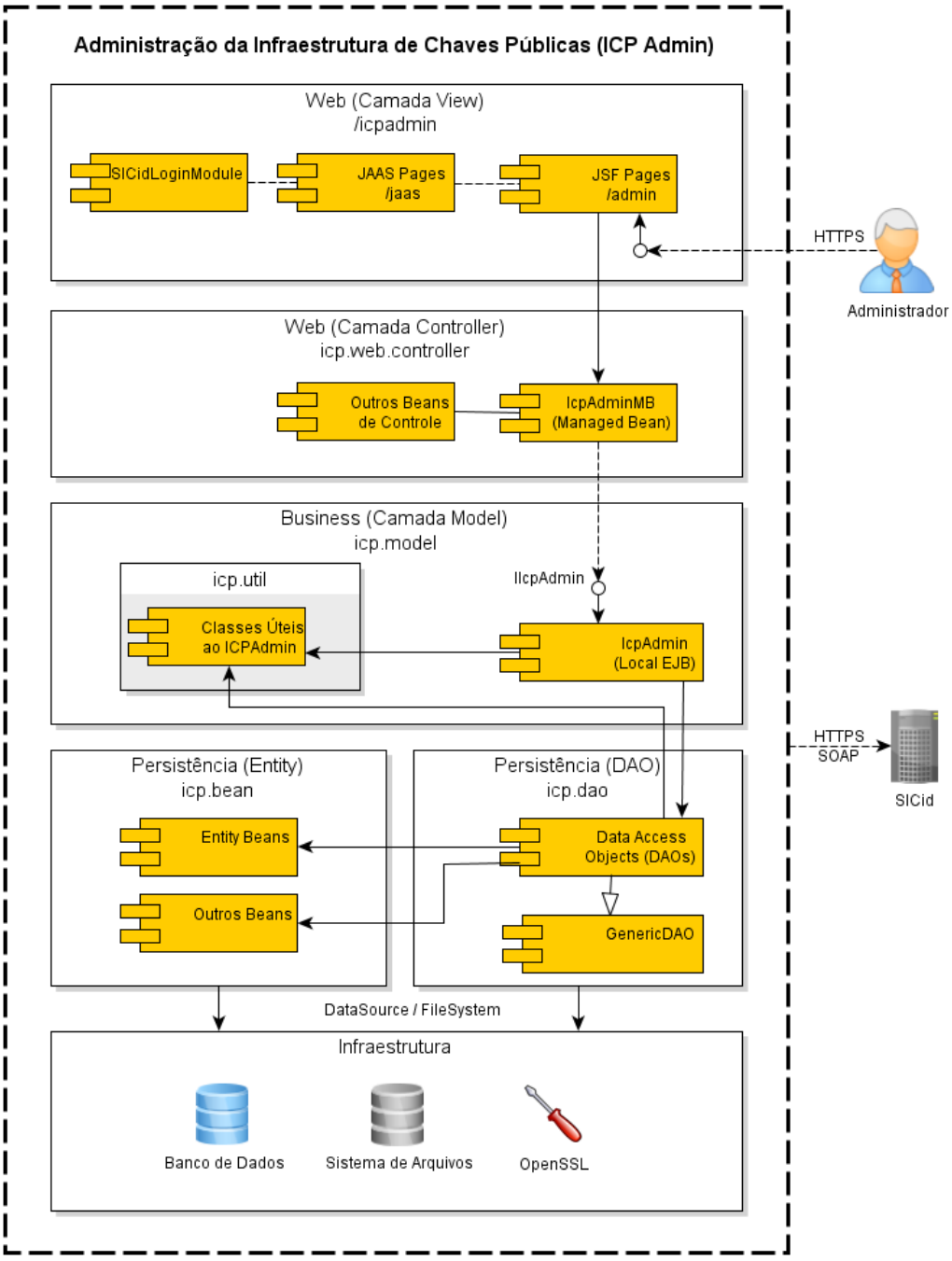

Figura 22 - Visão de implementação do sistema ICP Admin

3.3.6.1 Tecnologias e ferramentas utilizadas

As seguintes tecnologias estão relacionadas com a implementação desta prova de conceito:

Sigla	Nome	Descrição
SOA	Service-Oriented Architecture	"A Arquitetura Orientada a Serviços (SOA) é um paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários" (OASIS, 2006).
-	Cloud Computing	A Computação em Nuvem é uma forma de computação distribuída, onde aplicações de *software* são executadas em diferentes servidores fisicamente separados, mas logicamente reunidas como um serviço ("*as a service*") (VELTE, 2010).
-	Assinatura Digital	"A Assinatura Digital é um mecanismo digital utilizado para fornecer confiabilidade, tanto sobre autenticidade de um determinado documento eletrônico, como sobre o remetente do mesmo" (VOLPI, 2001).
		Protocolos usados: RSA, SHA1
-	Certificação Digital	A Certificação Digital é um mecanismo que garante que o dono de um par de chaves criptográficas é realmente quem ele afirma ser (SILVA et al., 2008).
		Protocolos usados: X509.3, PKCS#12, PKCS#10, PKCS#7, Padrões da ICP-Brasil
-	Smart Card	O *Smart Card* é um cartão de plástico, com dimensões semelhantes a um cartão magnético comum, dotado de um *chip* capaz de armazenar de dados sensíveis, chaves criptográficas e certificados digitais (RANKL, 2003).
		Protocolos usados: PKCS#11
SSL	Secure Sockets Layer	Protocolo que utiliza criptografia na camada de transporte da rede, visando garantir a confidencialidade das informações (DAVIE, PETERSON, 2007). Usado em conjunto com o *HyperText Transfer Protocol* (HTTP) resultando no protocolo conhecido como HTTPS.
JEE	Java Enterprise Edition	Plataforma para desenvolvimento de aplicações corporativas usando a arquitetura Java (ORACLE, 2012).
-	Web Service (SOAP)	"O *Web Service* é uma aplicação de *software* identificada por uma URI, onde interfaces são definidas, descritas e publicadas como artefatos XML" (W3C, 2004, tradução nossa).
		Tecnologias correlatas: XML, XSD, SOAP, WSDL, HTTP
JAX-WS	Java API for XML Web Services	JAX-WS é a especificação de uma *Application Programming Interface* (API) para a construção de *web services* usando Java (ORACLE, 2012).
JAXB	Java Architecture for XML Binding	JAXB é a especificação de uma API que permite a representação de objetos Java como elementos XML e vice-versa (ORACLE, 2012).

Sigla	Nome	Descrição
EJB	Enterprise Java Beans	O EJB é um componente de *software* que faz parte da especificação *Java Enterprise Edition* (JEE), e que essencialmente executa num contêiner (servidor) de aplicação (ORACLE, 2012).
JPA	Java Persistence API	JPA é a especificação de uma API que permite a persistência de objetos Java em bancos de dados (ORACLE, 2012). Esta API garante a funcionalidade conhecida por Mapeamento Objeto-Relacional (*Object Relational Mapping* - ORM).
-	Hibernate	Um *framework* destinado à persistência de objetos Java em bancos de dados (JBOSS, 2012). É uma das implementações mais conhecidas e utilizadas de JPA.
JAAS	Java Authentication and Authorization Service	JAAS é uma API Java que oferece serviços de autenticação e autorização a aplicações construídas no padrão Java EE (ORACLE, 2012).
JCA	Java Cryptography Architecture	Especificação de uma arquitetura destinada a oferecer funcionalidades criptográficas (incluindo manipulação de chaves, assinatura e certificação digital) a aplicações baseadas em Java (ORACLE, 2012).
JCE	Java Cryptography Extension	*Framework* que implementa a arquitetura JCA, incluído na distribuição Java (JSE) padrão.
JSF	Java Server Faces	Especificação (e *framework*) destinado a simplificar a construção de aplicações *web* baseadas em Java.
JSP	Java Server Pages	O JSP é uma tecnologia usada no desenvolvimento de aplicações *web* em Java, focada na elaboração de páginas dinâmicas processadas no servidor de aplicação.
-	Java Applet	Código baseado em Java capaz de ser executado num computador cliente, num contexto do navegador de *internet*.
-	Java Swing	*Framework* gráfico (*widget toolkit*), baseado em Java, que permite a elaboração de interfaces gráficas com o usuário (GUI).

Quadro 21 - Tecnologias relacionadas com a implementação

Os seguintes *frameworks*, bibliotecas e utilitários fazem parte da implementação:

Nome	Versão	Descrição
Bouncy Castle	1.4.7	Biblioteca criptográfica para Java e outras linguagens (BOUNCY CASTLE, 2012).
Ini4J	0.5.2	API simples para Java que permite manipular arquivos no formato *Windows* INI (INI4J, 2012).
PrimeFaces	3.3.1	Suíte de componentes visuais para *Java Server Faces* – JSF 2 (PRIMEFACES, 2011).
PrimeFaces Extensions	0.6.1	Componentes adicionais para *PrimeFaces* e JSF 2 (PRIMEFACES EXTENSIONS, 2012).
Apache Commons	FileUpload: 1.2.2 Commons IO: 2.4 Commons Lang: 3.1	Pacotes de componentes reutilizáveis para Java (APACHE, 2012).

Nome	Versão	Descrição
HyperSQL (HSQLDB)	Embarcado no servidor de aplicação JBoss	Sistema de banco de dados relacional, escrito completamente em Java (HSQL, 2012).
Log4J	Embarcado no servidor de aplicação JBoss	Biblioteca Java que provê serviços de *log* (APACHE, 2004).
OpenSSL	1.0	Um conjunto de ferramentas que implementam utilidades criptográficas (OPENSSL, 2009).

Quadro 22 - *Frameworks*, bibliotecas e utilitários usados na implementação

As seguintes ferramentas e ambientes auxiliaram no desenvolvimento dos sistemas que compõem esta prova de conceito, e na elaboração da documentação do projeto:

Nome	Versão	Descrição
Java SE JDK	7	A plataforma *Java Standard Edition* (Java SE) possibilita o desenvolvimento e implantação de aplicações baseadas em Java (ORACLE, 2012).
Eclipse JEE IDE	3.6.2 (Helios)	Ambiente Integrado de Desenvolvimento (IDE), que permite o desenvolvimento de aplicações baseadas em Java (ECLIPSE FOUNDATION, 2012).
JBoss Aplication Server	6.1 (Community)	Um servidor de aplicações *Java Enterprise Edition*, em edição *open-source* (JBOSS, 2012).
JBossTools (Eclipse plugin)	3.2	*Plugin* para o ambiente *Eclipse* que oferece funcionalidades relacionadas ao servidor de aplicação *JBoss* (JBOSS, 2012).
KeyTool	Integrado ao Java SE JDK	Utilitário de linha de comando que permite a manipulação de chaves criptográficas, certificados digitais e arquivos de *keystore*.
JarSigner	Integrado ao Java SE JDK	Utilitário de linha de comando que permite assinar uma *applet* Java usando um certificado digital.
JavaDoc	Integrado ao Java SE JDK	Utilitário de linha de comando que gera documentação HTML a partir de código-fonte Java.
Apache Ant	Integrado ao Eclipse IDE	Utilitário para a construção de aplicações Java (APACHE, 2003).
Portecle	1.7	"Utilitário gráfico amigável para criar, gerenciar e examinar chaves criptográficas, certificados digitais, listas de revogação (LCR), *keystores*, etc." (PORTECLE, 2004, tradução nossa).
SoapUI	4.5	Aplicação para realizar testes em *web services* baseados em SOAP/WSDL ou REST/WADL (SMARTBEAR, 2005).
IBM Rational Team Concert (RTC)	3.0.1.3 (Free 10 Developers)	O *IBM Rational Team Concert* (RTC) é uma solução colaborativa para o gerenciamento de ciclo de vida de aplicações de *software* (C*ollaborative Application Lifecycle Management* – CALM) (IBM, 2008).

Nome	Versão	Descrição
RTC Eclipse Plugin	3.0.1.3	*Plugin* para o ambiente *Eclipse* que permite usar funcionalidades do *Rational Team Concert*, como *load/deliver* de código-fonte, associação de mudanças a itens de trabalho, etc.
Oracle VM VirtualBox	4.1.18	Um pacote de *software* destinado à virtualização de sistemas baseados em processadores x86 (ORACLE, 2008).
Ubuntu Linux	12.04 LTS (Precise Pangolin)	*Ubuntu* é uma distribuição *GNU/Linux* baseada no *Debian* (CANONICAL, 2012).
yEd Graph Editor	3.9.2	Um poderoso e gratuito editor gráfico, que gera diagramas de alta qualidade (YWORKS, 2012).
LibreOffice	3.5.4	"O *LibreOffice* é uma suíte de escritório livre compatível com as principais suítes de escritório do mercado" (TDF, 2010).
GIMP	2.6.12	*GNU Image Manipulation Program* é uma aplicação *open-source* para manipulação de imagens e fotografias digitais (GIMP, 2001).
Mozilla Firefox	16.0.2	Navegador *web open-source* e multiplataforma (MOZILLA, 1998).

Quadro 23 - Ferramentas e ambientes usados no desenvolvimento

3.3.7 Visão de Implantação

O diagrama abaixo ilustra a implantação dos sistemas que compõem a prova de conceito deste trabalho.

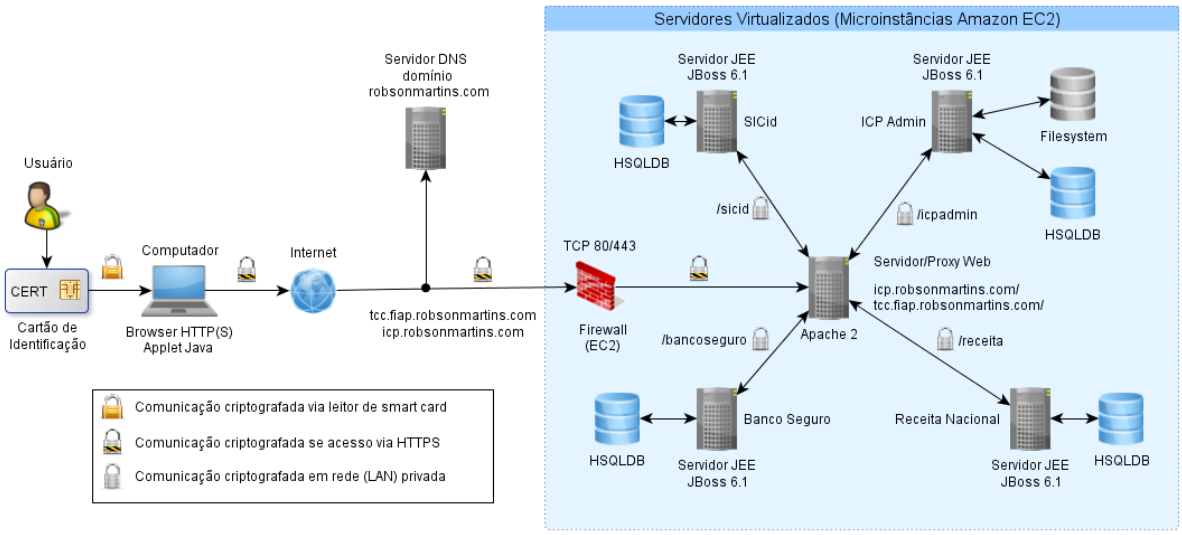

Figura 23 - Arquitetura de implantação da prova de conceito

3.3.7.1 Softwares e ferramentas utilizadas

Os seguintes *softwares*, serviços e ferramentas auxiliaram na implantação dos sistemas que compõem esta prova de conceito:

Nome / Versão	Categoria	Descrição
Amazon Elastic Compute Cloud (EC2) [Free 1 Year Trial]	Serviço de *Cloud Computing*	"O *Amazon Elastic Compute Cloud* (Amazon EC2) é um serviço da *Web* que fornece uma capacidade de computação redimensionável na nuvem" (AMAZON, 2012).
Ubuntu Server Linux 12.04 LTS (Precise Pangolin)	Sistema Operacional	*Ubuntu Server* é uma distribuição *GNU/Linux* baseada no *Debian*, destinada a servidores (CANONICAL, 2012).
Apache HTTP Server 2.2	Servidor *Web*	Um popular servidor *web open-source*, que oferece os principais serviços HTTP (APACHE, 1996).
Oracle Java JSE/JDK 7	Java *Runtime*	A plataforma *Java Standard Edition* (Java SE) possibilita o desenvolvimento e implantação de aplicações baseadas em Java (ORACLE, 2012).
JBoss Application Server 6.1 Community Edition	Servidor de Aplicação (JEE)	Um servidor de aplicações *Java Enterprise Edition*, em edição *open-source* (JBOSS, 2012).
OpenSSL 1.0	Utilitário de Linha de comando	Um conjunto de ferramentas que implementam utilidades criptográficas (OPENSSL, 2009).

Quadro 24 - *Softwares*, serviços e ferramentas na implantação dos sistemas

3.3.8 Visão de Integração

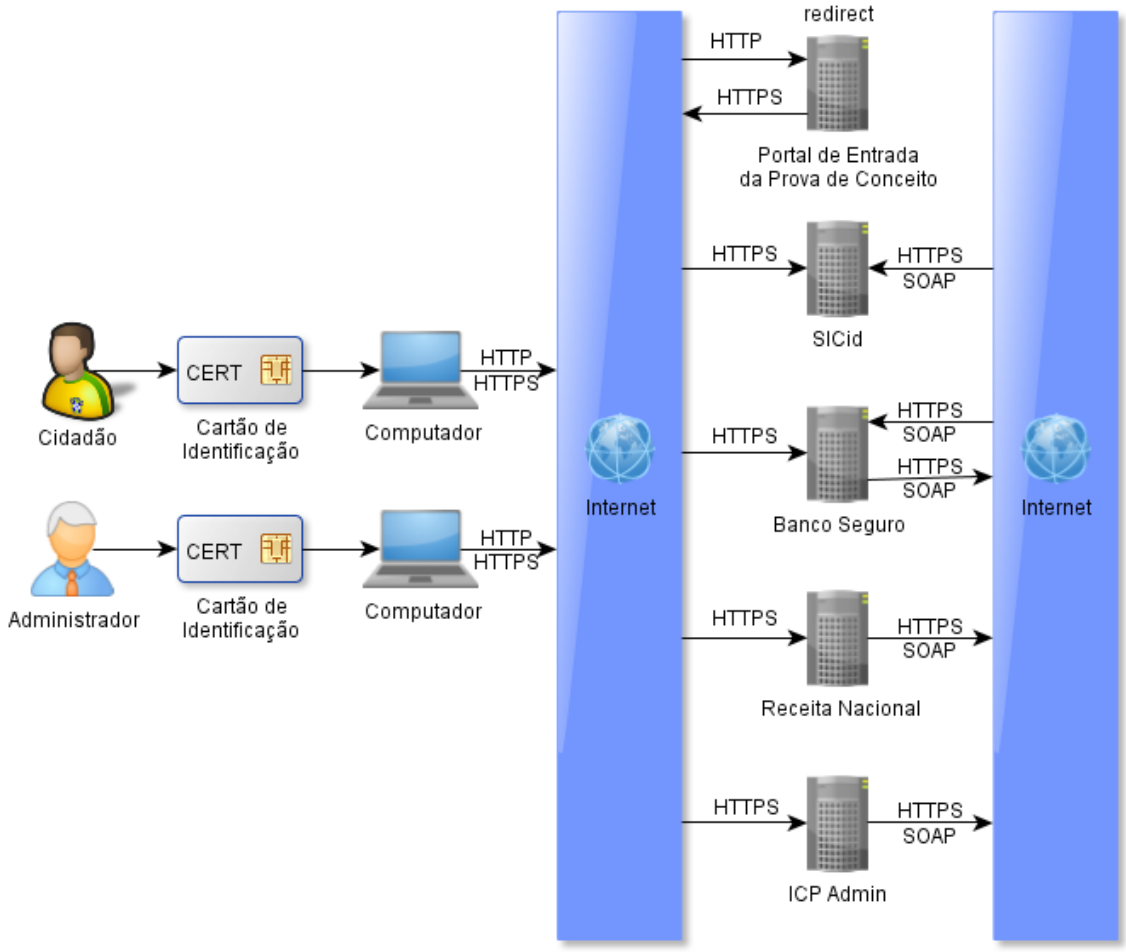

Figura 24 - Visão de integração da prova de conceito

Este diagrama ilustra a integração entre os sistemas que compõem a prova de conceito deste trabalho.

Nesta arquitetura de integração, os serviços interagem uns com os outros através de protocolo SOAP (W*eb Service* XML), sobre HTTPS, mediante autenticação e autorização (conforme detalhado na seção “Visão de Casos de Uso / Realização de Casos de Uso”, deste Documento de Arquitetura).

O acesso aos sistemas (aplicações *web*) também se dá por meio de protocolo HTTPS, mediante autenticação e autorização do usuário (cidadão, administrador ou outro perfil).

O “Portal de Entrada” é um *site* estático que contém *links* para cada uma das aplicações implementadas nesta prova de conceito, e algumas informações adicionais sobre o trabalho.

3.3.9 Segurança

Todos os sistemas (aplicações *web* e serviços) que compõem esta prova de conceito são acessíveis via *internet*. Por esse motivo, a implementação de autenticação (e autorização) são fundamentais para garantir os aspectos relacionados a segurança da solução.

O quadro a seguir apresenta os níveis de acesso (perfis de usuário) das aplicações e serviços que compõem esta prova de conceito.

Sistema / Serviço	Perfil	Descrição
SICid Admin	`admin` (Administrador)	Usuários com perfil de Administrador podem acessar e efetuar todas as operações da aplicação *web*.
	Outros	Nenhum acesso permitido.
Serviço SICid	`wsuser` (Aplicação Confiável)	Aplicações com perfil de Aplicação Confiável podem acessar o WSDL e consumir os serviços oferecidos.
	Outros	Nenhum acesso permitido.
Banco Seguro	`gerente` (Gerente)	Usuários com perfil de Gerente podem acessar a aplicação do Banco Seguro e efetuar as operações permitidas somente a Gerentes. Também podem ter o mesmo acesso do perfil Cliente, desde que possuam uma conta aberta no banco.
	`cliente` (Cliente)	Usuários com perfil de Cliente podem acessar a aplicação do Banco Seguro e efetuar operações de movimentação bancária de sua própria conta (serviço de *internet banking*).
	Outros	Nenhum acesso permitido. Podem visualizar somente uma página pública com uma mensagem semelhante a “Abra já a sua conta”.
Serviço do Banco Seguro	`wsuser` (Aplicação Confiável)	Aplicações com perfil de Aplicação Confiável podem acessar o WSDL e consumir os serviços oferecidos.
	Outros	Nenhum acesso permitido.
Receita Nacional	`admin` (Administrador ou Representante de Governo)	Usuários com perfil de Administrador (ou Representante de Governo) podem acessar a aplicação da Receita Nacional e efetuar as operações permitidas somente a Administradores. Também têm o mesmo acesso do perfil Cidadão.
	`cidadao` (Cidadão)	Usuários com perfil de Cidadão podem acessar a aplicação da Receita Nacional e efetuar operações relativas a sua própria situação tributária.
	Outros	Nenhum acesso permitido.
ICP Admin	`admin` (Administrador)	Usuários com perfil de Administrador podem acessar e efetuar todas as operações da aplicação *web*.
	Outros	Nenhum acesso permitido.

Quadro 25 - Perfis de usuário dos sistemas e serviços da prova de conceito

4 CÓDIGO-FONTE

Esta seção contém o código-fonte de alguns dos principais módulos dos sistemas e serviços que compõem a prova de conceito que demonstra este trabalho. Por questões de espaço disponível, não está transcrito aqui o todo o código-fonte da solução.

A documentação *JavaDoc* do código-fonte da prova de conceito está publicada e disponibilizada no endereço: <http://tcc.fiap.robsonmartins.com/javadoc> (acesso em: 12 nov. 2012).

4.1 *Applet* de Autenticação do Serviço SICid

SICidApplet.java

```
import java.awt.BorderLayout;
import java.awt.Cursor;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.security.AccessController;
import java.security.KeyStore;
import java.security.PrivilegedAction;
import java.security.Provider;
import java.security.cert.X509Certificate;
import java.util.List;
import javax.swing.DefaultListModel;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JDesktopPane;
import javax.swing.JFileChooser;
import javax.swing.JInternalFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.border.EtchedBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
```

SICidApplet.java

```
import javax.swing.filechooser.FileFilter;
import netscape.javascript.JSObject;
import sun.misc.BASE64Encoder;

/**
 * Applet para selecionar um certificado do cliente a partir de
 *   dispositivos criptograficos (tokens ou smart cards).<p/>
 * <b>Parametros:</b>
 * <ul>
 * <li><i>form</i>: ID do form onde a applet esta' inserida (na pagina HTML).</li>
 * <li><i>username</i>: Nome do campo username (default: j_username).</li>
 * <li><i>password</i>: Nome do campo password (default: j_password).</li>
 * </ul>
 * @author Robson Martins (robson@robsonmartins.com)
 */
@SuppressWarnings("serial")
public class SICidApplet extends JApplet {
    /* componentes 'swing' */
    private JDesktopPane deskPane;
    private JInternalFrame dlgProviders;
    @SuppressWarnings("rawtypes")
    private JList listProviders;
    @SuppressWarnings("rawtypes")
    private DefaultListModel listModelProviders;
    private JPanel btnProviderPanel;
    private JButton btnProviderOk;
    private JButton btnProviderRefresh;
    private JButton btnPkcs12;
    private JInternalFrame dlgCerts;
    @SuppressWarnings("rawtypes")
    private JList listCerts;
    @SuppressWarnings("rawtypes")
    private DefaultListModel listModelCerts;
    private JPanel btnCertPanel;
    private JButton btnCertOk;
    private JButton btnCertCancelar;
    /* lista de providers disponiveis */
    private List<Provider> providers;
    /* lista de certificados presentes */
    private List<KeyStore.PrivateKeyEntry> certs;
    /* provider selecionado */
    private Provider selectedProvider;
    /* certificado selecionado */
    private KeyStore.PrivateKeyEntry selectedCert;
    /* objetos para interacao com o browser (JavaScript) */
    private JSObject browserWindow;
    private JSObject mainForm;
```

SICidApplet.java

```
    private JSObject usernameField;
    private JSObject passwordField;
    /* id do form onde a applet esta' inserida */
    private String formId;
    /* nome dos campos de autenticacao */
    private String usernameFieldName;
    private String passwordFieldName;

    /**
     * Inicializa a applet.
     */
    @Override
    public void init() {
        try {
            formId            = getParameter("form"    );
            usernameFieldName = getParameter("username");
            passwordFieldName = getParameter("password");
            browserWindow = JSObject.getWindow(this);
           if (formId != null && !"".equals(formId)) {
                mainForm = (JSObject) browserWindow.eval(
                    String.format("document.getElementById('%s')",
                            formId));
            } else {
                mainForm = (JSObject)
                    browserWindow.eval("document.forms[0]");
            }
            if (usernameFieldName == null || "".equals(usernameFieldName)){
                usernameFieldName = "j_username";
            }
            if (passwordFieldName == null || "".equals(passwordFieldName)){
                passwordFieldName = "j_password";
            }
            usernameField = (JSObject) mainForm.getMember(usernameFieldName);
            passwordField = (JSObject) mainForm.getMember(passwordFieldName);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,"Erro ao iniciar applet.",
                    "SICid Login Applet", JOptionPane.ERROR_MESSAGE);
        }
        try {
            deskPane = new JDesktopPane();
            add(deskPane);
            createDlgProviders();
            createDlgCerts();
            deskPane.add(dlgProviders);
            deskPane.add(dlgCerts);
            dlgProviders.setMaximum(true);
            dlgCerts.setMaximum(true);
```

SICidApplet.java

```
            dlgProviders.setVisible(true);
            dlgCerts.setVisible(false);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,"Erro ao exibir janela.",
                    "SICid Login Applet", JOptionPane.ERROR_MESSAGE);
        }
        refreshProviderList();
    }
    /* Cria o dialogo de selecao de provider. */
    @SuppressWarnings({ "rawtypes", "unchecked" })
    private void createDlgProviders() {
        dlgProviders = new JInternalFrame("Selecione o Dispositivo");
        dlgProviders.setLayout(new BorderLayout(0, 0));
        listModelProviders = new DefaultListModel();
        listProviders = new JList();
        listProviders.setBorder(
            new EtchedBorder(EtchedBorder.LOWERED, null, null));
        listProviders.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        listProviders.setModel(listModelProviders);
        listProviders.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                btnProviderOk.setEnabled(
                    listProviders.getSelectedIndex() >= 0);
            }
        });
        listProviders.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e){
                if (e.getClickCount() == 2){
                    int idx = listProviders.getSelectedIndex();
                    if (idx >= 0) {
                        selectedProvider = providers.get(idx);
                        dlgProviders.setVisible(false);
                        dlgCerts.setVisible(true);
                        refreshCertListFromProvider();
                    } else {
                        selectedProvider = null;
                    }
                }
            }
        });
        dlgProviders.add(new JScrollPane(listProviders), BorderLayout.CENTER);
        btnProviderPanel = new JPanel();
        dlgProviders.add(btnProviderPanel, BorderLayout.SOUTH);
        btnProviderPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        btnProviderOk = new JButton("OK");
```

SICidApplet.java

```
        btnProviderOk.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int idx = listProviders.getSelectedIndex();
                if (idx >= 0) {
                    selectedProvider = providers.get(idx);
                } else {
                    selectedProvider = null;
                }
                dlgProviders.setVisible(false);
                dlgCerts.setVisible(true);
                refreshCertListFromProvider();
            }
        });
        btnProviderOk.setEnabled(false);
        btnProviderPanel.add(btnProviderOk);
        btnProviderRefresh = new JButton("Recarregar");
        btnProviderRefresh.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                refreshProviderList();
            }
        });
        btnProviderPanel.add(btnProviderRefresh);
        btnPkcs12 = new JButton("Arquivo");
        btnPkcs12.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dlgProviders.setVisible(false);
                dlgCerts.setVisible(true);
                refreshCertListFromPkcs12();
                selectedProvider = null;
            }
        });
        btnProviderPanel.add(btnPkcs12);
    }

    /* Cria o dialogo de selecao de certificado. */
    @SuppressWarnings({ "rawtypes", "unchecked" })
    private void createDlgCerts() {
        dlgCerts = new JInternalFrame("Selecione o Certificado");
        dlgCerts.setLayout(new BorderLayout(0, 0));
        listModelCerts = new DefaultListModel();
        listCerts = new JList();
        listCerts.setBorder(
            new EtchedBorder(EtchedBorder.LOWERED, null, null));
        listCerts.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        listCerts.setModel(listModelCerts);
```

SICidApplet.java

```
        listCerts.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                btnCertOk.setEnabled(listCerts.getSelectedIndex() >= 0);
            }
        });
        listCerts.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e){
                if (e.getClickCount() == 2){
                    int idx = listCerts.getSelectedIndex();
                    if (idx >= 0) {
                        selectedCert = certs.get(idx);
                        dlgProviders.setVisible(false);
                        dlgCerts.setVisible(false);
                        postCertificate();
                    } else {
                        selectedCert = null;
                    }
                }
            }
        });
        dlgCerts.add(new JScrollPane(listCerts), BorderLayout.CENTER);
        btnCertPanel = new JPanel();
        dlgCerts.add(btnCertPanel, BorderLayout.SOUTH);
        btnCertPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        btnCertOk = new JButton("OK");
        btnCertOk.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int idx = listCerts.getSelectedIndex();
                if (idx >= 0) {
                    selectedCert = certs.get(idx);
                    dlgProviders.setVisible(false);
                    dlgCerts.setVisible(false);
                    postCertificate();
                } else {
                    selectedCert = null;
                }
            }
        });
        btnCertOk.setEnabled(false);
        btnCertPanel.add(btnCertOk);
        btnCertCancelar = new JButton("Voltar");
```

SICidApplet.java

```
        btnCertCancelar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dlgCerts.setVisible(false);
                dlgProviders.setVisible(true);
                selectedCert = null;
            }
        });
        btnCertCancelar.setEnabled(false);
        btnCertPanel.add(btnCertCancelar);
    }

    /* Carrega o conjunto de providers presentes no sistema. */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    private void loadProviders() {
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                try {
                    providers = CryptoDeviceManager.getProviderList();
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null,
                        "Erro ao carregar lista de dispositivos.",
                        "SICid Login Applet",
                        JOptionPane.ERROR_MESSAGE);
                    providers = null;
                }
                return null;
            }
        });
    }

    /* Carrega o conjunto de certificados presentes no dispositivo. */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    private void loadCertsFromProvider() {
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                try {
                    certs = CryptoDeviceManager.getCertificates(selectedProvider);
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null,
                          "Erro ao carregar lista de certificados.",
                          "SICid Login Applet",
                          JOptionPane.ERROR_MESSAGE);
                    certs = null;
                }
                return null;
            }
        });
```

SICidApplet.java

```
}

/* Carrega o conjunto de certificados presentes em um arquivo PKCS#12. */
@SuppressWarnings({ "unchecked", "rawtypes" })
private void loadCertsFromPkcs12() {
    AccessController.doPrivileged(new PrivilegedAction() {
        public Object run() {
            try {
                File ksFile = selectPkcsFile();
                if (ksFile == null) { return null; }
                certs = CryptoDeviceManager.getCertificatesFromFile(ksFile);
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null,
                    "Erro ao carregar lista de certificados.",
                    "SICid Login Applet",
                    JOptionPane.ERROR_MESSAGE);
                certs = null;
            }
            return null;
        }
    });
}

/* Exibe um dialogo de selecao e retorna um arquivo PKCS#12.
 * @return Objeto {@link File} representando o arquivo PKCS#12 selecionado. */
private File selectPkcsFile() {
    JFileChooser fileDlg = new JFileChooser();
    FileFilter filter = new FileFilter() {
        @Override
        public String getDescription() {
            return "Arquivos de certificado PKCS#12 (*.pfx,*.p12)";
        }
        @Override
        public boolean accept(File f) {
            return (f.isDirectory()
                    || f.getAbsolutePath().toLowerCase().endsWith(".pfx")
                    || f.getAbsolutePath().toLowerCase().endsWith(".p12"));
        }
    };
    fileDlg.setFileFilter(filter);
    if (fileDlg.showOpenDialog(deskPane) == JFileChooser.APPROVE_OPTION){
        return fileDlg.getSelectedFile();
    } else {
        return null;
    }
}
```

SICidApplet.java

```
    /* Recarrega a lista de providers. */
    @SuppressWarnings("unchecked")
    private void refreshProviderList() {
        btnProviderOk.setEnabled(false);
        btnProviderRefresh.setEnabled(false);
        btnPkcs12.setEnabled(false);
        listModelProviders.clear();
        dlgProviders.paintImmediately(dlgProviders.getBounds());
        selectedProvider = null;
        dlgProviders.setCursor(new Cursor(Cursor.WAIT_CURSOR));
        loadProviders();
        dlgProviders.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        if (providers != null) {
            for (Provider p : providers) {
                listModelProviders.addElement(p.toString());
            }
        }
        listProviders.setSelectedIndex((listModelProviders.size()) > 0 ? 0 : -1);
        btnProviderOk.setEnabled(listProviders.getSelectedIndex() >= 0);
        btnProviderRefresh.setEnabled(true);
        btnPkcs12.setEnabled(true);
    }

    /* Recarrega a lista de certificados,
     * a partir de um dispositivo */
    private void refreshCertListFromProvider() {
        refreshCertList(false);
    }

    /* Recarrega a lista de certificados,
     * a partir de um arquivo PKCS#12 */
    private void refreshCertListFromPkcs12() {
        refreshCertList(true);
    }

    /* Recarrega a lista de certificados,
     * @param pkcs12 Se true, indica que a lista sera'
     *   obtida a partir de um arquivo PKCS#12. Se false,
     *   a lista sera' obtida a partir de um provider. */
    @SuppressWarnings("unchecked")
    private void refreshCertList(boolean pkcs12) {
        btnCertOk.setEnabled(false);
        btnCertCancelar.setEnabled(false);
        btnPkcs12.setEnabled(false);
        listModelCerts.clear();
        dlgCerts.paintImmediately(dlgCerts.getBounds());
        selectedCert = null;
```

SICidApplet.java

```
        dlgCerts.setCursor(new Cursor(Cursor.WAIT_CURSOR));
        if (pkcs12) {
            loadCertsFromPkcs12();
        } else {
            loadCertsFromProvider();
        }
        dlgCerts.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        if (certs != null) {
            for (KeyStore.PrivateKeyEntry entry : certs) {
                X509Certificate cert =
                    (X509Certificate) entry.getCertificate();
                List<String> listCN =
                    CryptoDeviceManager.subjectParse(
                        cert.getSubjectX500Principal().getName(),
                        "CN");
                if (listCN.size() != 0) {
                    listModelCerts.addElement(listCN.get(0));
                } else {
                    listModelCerts.addElement("Unknown");
                }
            }
        }
        listCerts.setSelectedIndex((listModelCerts.size()) > 0 ? 0 : -1);
        btnCertOk.setEnabled(listCerts.getSelectedIndex() >= 0);
        btnCertCancelar.setEnabled(true);
        btnPkcs12.setEnabled(true);
    }

    /* Preenche os campos do form e realiza o submit */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    private void postCertificate() {
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                if (selectedCert != null) {
                    try {
                        X509Certificate cert = (X509Certificate)
                            selectedCert.getCertificate();
                        String username = CertificadoSerializador.certToStr(cert);
                        String password = new BASE64Encoder().encode(
                            CertificadoAssinador.sign(selectedProvider,
                                "SHA1withRSA", selectedCert.getPrivateKey(),
                                cert.getEncoded()));
                        usernameField.setMember("value", username);
                        passwordField.setMember("value", password);
                        browserWindow.eval(String.format(
                            "document.getElementById('%s').submit();", formId));
```

SICidApplet.java

```
                    } catch (Exception e) {
                        JOptionPane.showMessageDialog(null,
                            "Erro ao submeter o certificado digital.",
                            "SICid Login Applet",
                            JOptionPane.ERROR_MESSAGE);
                    }
                }
                return null;
            }
        });
    }
}
```

CryptoDeviceType.java

```
/**
 * Enumera os Tipos de Dispositivos Criptograficos compativeis com esta applet.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public enum CryptoDeviceType {
    A3_ACOS5           ("ACOS5"           ,"acospkcs11"       ),
    A3_ATHENA          ("ATHENA"          ,"asepkcs"          ),
    A3_IDPROTECT       ("IDPROTECT"       ,"ASEP11"           ),
    A3_STARCOS         ("STARCOS"         ,"aetpkss1"         ),
    A3_ETOKEN          ("ETOKEN"          ,"etpkcs11"         ),
    A3_SAFEWEB         ("SAFEWEB"         ,"cmp11"            ),
    A3_EPASS1000       ("EPASS1000"       ,"ep1pk111"         ),
    A3_EPASS2000       ("EPASS2000"       ,"ep2pk11"          ),
    A3_EPASS3000       ("EPASS3000"       ,"ngp11v211"        ),
    A3_EPASS3003       ("EPASS3003"       ,"shuttlecsp11_3003"),
    A3_EPASS2000LX     ("EPASS2000LX"     ,"epsng_p11"        ),
    A3_AR_MINIKEY      ("ARMINIKEY"       ,"sadaptor"         ),
    A3_ALOAHA          ("ALOAHA"          ,"aloaha_pkcs11"    ),
    A3_ASIGN           ("ASIGN"           ,"psepkcs11"        ),
    A3_ATRUST          ("ATRUST"          ,"asignp11"         ),
    A3_CHRYSALIS       ("CHRYSALIS"       ,"cryst32"          ),
    A3_LUNA            ("LUNA"            ,"cryst201"         ),
    A3_IBUTTON         ("IBUTTON"         ,"dspkcs"           ),
    A3_ERACOM          ("ERACOM"          ,"cryptoki"         ),
    A3_GEMPLUS         ("GEMPLUS"         ,"gclib"            ),
    A3_GEMPLUS2        ("GEMPLUS2"        ,"pk2priv"          ),
    A3_GEMSOFT         ("GEMSOFT"         ,"w32pk2ig"         ),
    A3_IBM_DSI         ("IBM_DSI"         ,"cccsigit"         ),
    A3_IBM_ESS         ("IBM_ESS"         ,"csspkcs11"        ),
    A3_IBM_PSG         ("IBM_PSG"         ,"ibmpkcss"         ),
    A3_ID2             ("ID2"             ,"id2cbox"          ),
    A3_NFAST           ("NFAST"           ,"cknfast"          ),
    A3_NEXUS           ("NEXUS"           ,"nxpkcs11"         ),
    A3_MIRCADO         ("MIRCADO"         ,"micardopkcs11"    ),
    A3_CRYPTOSWIFT     ("CRYPTOSWIFT"     ,"cryptoki22"       ),
    A3_CRYPTOSWIFT_HSM("CRYPTOSWIFT_HSM","iveacryptoki"     ),
    A3_IKEY1000        ("IKEY1000"        ,"k1pk112"          ),
    A3_IKEY2000        ("IKEY2000"        ,"dkck201"          ),
    A3_IKEY2032        ("IKEY2032"        ,"dkck232"          ),
    A3_SAFELAYER_HSM   ("SAFELAYER_HSM"   ,"p11card"          ),
    A3_CRYPTOFLEX      ("CRYPTOFLEX"      ,"acpkcs"           ),
    A3_CRYPTOFLEX2     ("CRYPTOFLEX2"     ,"slbck"            ),
    A3_SETOKI          ("SETOKI"          ,"settoki"          ),
    A3_HIPATH          ("HIPATH"          ,"siecap11"         ),
    A3_SMARTTRUST      ("SMARTTRUST"      ,"smartp11"         ),
    A3_SPYRUS          ("SPYRUS"          ,"spypk11"          ),
    A3_UTIMACO         ("UTIMACO"         ,"pkcs201n"         ),
```

CryptoDeviceType.java

```
    A3_ACTIVCLIENT    ("ACTIVCLIENT"    ,"acpkcs211"        ),
    A3_FORTEZZA       ("FORTEZZA"       ,"fort32"           ),
    A3_AUTHENTIC      ("AUTHENTIC"      ,"aucryptoki2-0"    ),
    A3_SCW_3GI        ("SCW_3GI"        ,"3gp11csp"         ),
    A3_TELESEC        ("TELESEC"        ,"pkcs11"           ),
    A3_OPENSC         ("OPENSC"         ,"opensc-pkcs11"    );

    /* Nome do dispositivo. */
    private final String name;
    /* Nome da biblioteca */
    private final String library;

    /* Cria uma instancia de CryptoDeviceType.
     * @param name Nome do dispositivo.
     * @param library Nome da biblioteca.
     */
    private CryptoDeviceType(String name, String library) {
        this.name    = name;
        this.library = library;
    }

    /**
     * Retorna o nome do dispositivo.
     * @return Nome do dispositivo.
     */
    public String getName() {
        return name;
    }

    /**
     * Retorna o nome da biblioteca.
     * @return Nome da biblioteca.
     */
    public String getLibrary() {
        return library;
    }
}
```

CryptoDeviceManager.java

```
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.Provider;
import java.security.ProviderException;
import java.security.Security;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import javax.security.auth.callback.CallbackHandler;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import sun.security.pkcs11.SunPKCS11;
import com.sun.security.auth.callback.DialogCallbackHandler;

/**
 * Gerencia os Dispositivos Criptograficos.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class CryptoDeviceManager {
    /**
     * Retorna uma lista de Certificados presentes dentro do Dispositivo
     * Criptografico.
     * @param provider Provider de um Dispositivo Criptografico.
     * @return Lista de Certificados.
     * @throws Exception Se houve erro na obtencao dos certificados.
     */
    public static List<KeyStore.PrivateKeyEntry> getCertificates(
            Provider provider) throws Exception {

        List<KeyStore.PrivateKeyEntry> certList =
          new ArrayList<KeyStore.PrivateKeyEntry>();
        KeyStore ks = instanceOfKeyStore(provider);
        KeyStore.PrivateKeyEntry pkEntry = null;
        Enumeration<String> aliasesEnum = ks.aliases();
        while (aliasesEnum.hasMoreElements()) {
            String alias = (String) aliasesEnum.nextElement();
            if (ks.isKeyEntry(alias)) {
                pkEntry = (KeyStore.PrivateKeyEntry) ks.getEntry(alias,    null);
                certList.add(pkEntry);
            }
        }
        removeProvider(provider);
```

CryptoDeviceManager.java

```
        return certList;
    }

    /**
     * Autodetecta e retorna uma lista com os Providers dos
     * Dispositivos Criptograficos presentes no sistema.
     * @return Lista de Providers.
     * @throws Exception Se houve erro na autodeteccao de dispositivos.
     */
    public static List<Provider> getProviderList() throws Exception {
        List<Provider> providers = new ArrayList<Provider>();
           for (CryptoDeviceType d: CryptoDeviceType.values()) {
               Provider p = null;
               String libName = null;
               String deviceName = d.getName();
               try {
                   libName = getDeviceLibName(d);
                   if (libName != null) {
                       p = new SunPKCS11(getDeviceConfig(deviceName, libName));
                       providers.add(p);
                   }
               } catch (ProviderException e) {
                   continue;
               }
           }
           return providers;
    }

    /**
     * Libera os recursos de sistema alocados por um provider.
     * @param provider Provider de um Dispositivo Criptografico.
     */
    public static void removeProvider(Provider provider) {
        Security.removeProvider(provider.getName());
    }

    /**
     * Retorna uma lista de Certificados presentes
     * dentro de um arquivo de keystore.
     * @param keyStoreFile Arquivo de keystore.
     * @return Lista de Certificados.
     * @throws Exception Se houve erro na obtencao dos certificados.
     */
    public static List<KeyStore.PrivateKeyEntry>
                            getCertificatesFromFile(File keyStoreFile)
                                                    throws Exception {
```

CryptoDeviceManager.java

```
        List<KeyStore.PrivateKeyEntry> certList =
            new ArrayList<KeyStore.PrivateKeyEntry>();
        JPasswordField passField = new JPasswordField();
        JLabel passLabel = new JLabel();
        passField.setText(null);
        passLabel.setText("Enter the keystore password:");
        JOptionPane.showConfirmDialog(null, new Object[]{ passLabel, passField },
                "Load PKCS#12 Keystore", JOptionPane.OK_CANCEL_OPTION);
        KeyStore ks =
            instanceOfKeyStoreFromFile(keyStoreFile, passField.getPassword());
        KeyStore.PrivateKeyEntry pkEntry = null;
        Enumeration<String> aliasesEnum = ks.aliases();
        while (aliasesEnum.hasMoreElements()) {
            String alias = (String) aliasesEnum.nextElement();
            if (ks.isKeyEntry(alias)) {
                passField.setText(null);
                passLabel.setText(String.format(
                    "Enter the token [%s] password:",alias));
                JOptionPane.showConfirmDialog(null,
                        new Object[]{ passLabel, passField },
                        "Load PKCS#12 Keystore", JOptionPane.OK_CANCEL_OPTION);
                KeyStore.ProtectionParameter protParam =
                    new KeyStore.PasswordProtection(passField.getPassword());
                try {
                    pkEntry = (KeyStore.PrivateKeyEntry)
                        ks.getEntry(alias, protParam);
                    certList.add(pkEntry);
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null,"Incorrect password.",
                            "Load PKCS#12 Keystore", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
        return certList;
    }

    /**
     * Retorna os valores de um campo (field) de um subject
     * (DN - Distinguished Name).
     * @param subject String representando um subject.
     * @param field Nome do campo a ser retornado (ex: "C", "O", "OU", etc.).
     * @return Valores atribuidos ao campo, ou vazio se nenhum.
     * @see <a href="http://www.x500standard.com/">X500 Standard</a>
     */
    public static List<String> subjectParse(String subject, String field) {
        List<String> subjList = new ArrayList<String>();
        if (subject != null && !"".equals(subject)) {
```

CryptoDeviceManager.java

```
            String pattern = String.format("%s=", field);
            int beginIndex = 0;
            int endIndex = 0;
            do {
                beginIndex = subject.indexOf(pattern, beginIndex);
                if (beginIndex >= 0) {
                    beginIndex += pattern.length();
                    endIndex = subject.indexOf(",", beginIndex);
                    if (endIndex > - 0)
                        subjList.add(subject.substring(beginIndex, endIndex));
                    else
                        subjList.add(subject.substring(beginIndex));
                }
            } while (beginIndex >= 0);
        }
        return subjList;
    }

    /* Retorna uma instancia de KeyStore para o Provider especificado.
     * @param provider Provider de um Dispositivo Criptografico.
     * @return Instancia de KeyStore.
     * @throws Exception Se houve erro ao obter o KeyStore do Dispositivo.
     */
    private static KeyStore instanceOfKeyStore(Provider provider)
                                    throws Exception {
        KeyStore keyStore = null;
        Security.addProvider(provider);
        CallbackHandler cmdLineHdlr = new DialogCallbackHandler();
        KeyStore.Builder builder = KeyStore.Builder.newInstance ("PKCS11", null,
                    new KeyStore.CallbackHandlerProtection(cmdLineHdlr));
        try {
            keyStore = builder.getKeyStore();
        } catch (KeyStoreException e) {
            throw new Exception("Incorrect password or invalid certificate.");
        }
        return keyStore;
    }

    /* Retorna uma instancia de KeyStore para o Arquivo especificado.
     * @param keyStoreFile Arquivo de keystore.
     * @param password Senha do arquivo de keystore.
     * @return Instancia de KeyStore.
     * @throws Exception Se houve erro ao obter o KeyStore do Arquivo.
     */
    private static KeyStore instanceOfKeyStoreFromFile(File keyStoreFile,
            char[] password) throws Exception {
```

CryptoDeviceManager.java

```
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        InputStream istream = new FileInputStream(keyStoreFile);
        keyStore.load(istream, password);
        return keyStore;
    }

    /* Retorna uma stream com o conteudo do arquivo de configuracao gerado para o
     *   Dispositivo Criptografico especificado.
     * @param deviceName Nome do dispositivo criptografico.
     * @param libName Nome (e caminho) da biblioteca PKCS#11 do dispositivo.
     * @return Conteudo do arquivo de configuracao.
     * @throws Exception Se houve erro na geracao do arquivo.
     */
    private static InputStream getDeviceConfig(String deviceName,
            String libName) throws Exception {

        StringBuilder conf = new StringBuilder();
        conf.append("name = ")
            .append(deviceName)
            .append("\n")
            .append("library = ")
            .append(libName)
            .append("\n")
            .append("showInfo = true");
        return new ByteArrayInputStream(conf.toString().getBytes("UTF-8"));
    }

    /* Retorna o nome e o caminho da biblioteca PKCS#11 do dispositivo
     *   criptografico especificado.
     * @param device Dispositivo Criptografico.
     * @return Nome da biblioteca PKCS#11.
     */
    private static String getDeviceLibName(CryptoDeviceType device) {
        StringBuilder libName = null;
        String os = System.getProperty("os.name");
        File libFile = null;
        if (os == null || os.toLowerCase().startsWith("windows")) {
            libName = new StringBuilder(System.getenv("windir"));
            libName.append("\\system32\\")
                    .append(device.getLibrary())
                    .append(".dll");
            libFile = new File(libName.toString());
            if (!libFile.exists()) { libName = null; }
        } else {
            String paths[] =
                { "/usr/lib/lib", "/lib/lib", "/usr/lib/", "/lib/" };
            for (String path : paths) {
```

CryptoDeviceManager.java

```
                libName = new StringBuilder(path);
                libName.append(device.getLibrary())
                    .append(".so");
                libFile = new File(libName.toString());
                if (libFile.exists()) {
                    break;
                } else {
                    libName = null;
                }
            }
        }
        return (libName != null) ? libName.toString() : null;
    }
}
```

com.robsonmartins.fiap.tcc.util.CertificadoSerializador.java

```
package com.robsonmartins.fiap.tcc.util;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

/**
 * Serializa um objeto certificado ({@link X509Certificate})
 *   no formato Base64.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class CertificadoSerializador {
    /* encoder Base64 */
    private static BASE64Encoder encoder = new BASE64Encoder();
    /* decoder Base64 */
    private static BASE64Decoder decoder = new BASE64Decoder();

    /**
     * Serializa um certificado para string (Base64).
     * @param cert Objeto que representa um certificado.
     * @return String codificada em Base64.
     * @throws Exception
     */
    public static String certToStr(X509Certificate cert) throws Exception {
        String out = null;
        if (cert != null) {
            out = encoder.encode(cert.getEncoded()).replaceAll("\\s+","");
        }
        return out;
    }

    /**
     * Desserializa um certificado a partir de uma string (Base64).
     * @param s String codificada em Base64.
     * @return Objeto que representa um certificado.
     * @throws Exception
     */
    public static X509Certificate strToCert(String s) throws Exception {
        CertificateFactory certificateFactory = null;
        X509Certificate out = null;
        if (s != null) {
            certificateFactory = CertificateFactory.getInstance("X.509");
```

com.robsonmartins.fiap.tcc.util.CertificadoSerializador.java

```
            ByteArrayInputStream byteInStream =
                new ByteArrayInputStream(
                    decoder.decodeBuffer(s.replaceAll("\\s+","")));
            out = (X509Certificate)
                certificateFactory.generateCertificate(byteInStream);
        }
        return out;
    }

    /**
     * Serializa um certificado para um array de bytes.
     * @param cert Objeto que representa um certificado.
     * @return Array de bytes.
     * @throws Exception
     */
    public static byte[] certToBytes(X509Certificate cert) throws Exception {
        byte[] out = null;
        if (cert != null) {
            out = cert.getEncoded();
        }
        return out;
    }

    /**
     * Desserializa um certificado a partir de um array de bytes.
     * @param buffer Array de bytes.
     * @return Objeto que representa um certificado.
     * @throws Exception
     */
    public static X509Certificate bytesToCert(byte[] buffer) throws Exception {
        CertificateFactory certificateFactory = null;
        X509Certificate out = null;
        if (buffer != null) {
            certificateFactory = CertificateFactory.getInstance("X.509");
            ByteArrayInputStream byteInStream =
                new ByteArrayInputStream(buffer);
            out = (X509Certificate)
                certificateFactory.generateCertificate(byteInStream);
        }
        return out;
    }

    /**
     * Retorna o Distinguished Name (DN) de um certificado.
     * @param content Conteudo do certificado, codificado em Base64.
     * @return DN do certificado
     */
```

com.robsonmartins.fiap.tcc.util.CertificadoSerializador.java

```
public static String getDNByCertStr(String content) throws Exception {
    X509Certificate cert = strToCert(content);
    return cert.getSubjectX500Principal().getName();
}

/**
 * Obtem um objeto certificado ({@link X509Certificate}) a partir de
 *   uma instancia de {@link InputStream}, que pode apontar para um
 *   arquivo de certificado (formatos PEM ou DER).
 * @param istream Objeto InputStream que aponta para um arquivo
 *   de certificado.
 * @return Objeto que representa um certificado.
 * @throws Exception
 */
public static X509Certificate loadCertFromStream(InputStream istream)
                                    throws Exception {

    InputStream filteredStream = filterPemCert(istream);
    CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
    X509Certificate x509cert =
        (X509Certificate)certFactory.generateCertificate(filteredStream);
    return x509cert;
}

/*
 * Filtra o conteudo de um certificado (PEM, Base64), eliminando tudo
 *   o que estiver entre o delimitador ("-----BEGIN CERTIFICATE-----").
 *
 * Isso permite a importacao de arquivos PEM gerados pelo OpenSSL,
 *   incompativeis com o JCA.
 *
 * @param istream Objeto InputStream de entrada, com o conteudo do
 *   certificado, binario (DER) ou Base64 (PEM).
 * @return Objeto InputStream com o conteudo do certificado,
 *   binario (DER) ou Base64 (PEM).
 * @throws Exception
 */
private static InputStream filterPemCert(InputStream istream)
        throws Exception {

    /* delimitador de inicio do conteudo do certificado */
    final byte[] begin = "-----BEGIN CERTIFICATE-----".getBytes();
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    byte[] buffer = new byte[4 * 1024];
    int offset = 0;
    int len = 0;
```

com.robsonmartins.fiap.tcc.util.CertificadoSerializador.java

```
        /* copia todo conteudo da inputStream de entrada para um
         * buffer interno */
        while (-1 != (len = istream.read(buffer))) {
            output.write(buffer, 0, len);
        }
        buffer = output.toByteArray();
        /* vertifica tamanho do buffer */
        offset = 0;
        len = buffer.length;
        if (buffer.length < begin.length) {
            throw new Exception("Invalid input.");
        }
        /* procura pelo delimitador de inicio de certificado */
        int idxBuffer = 0;
        while (idxBuffer < buffer.length) {
            boolean found = true;
            for (int idxBegin = 0; idxBegin < begin.length; idxBegin++) {
                if (buffer[idxBuffer] != begin[idxBegin]) {
                    found = false;
                    break;
                }
                idxBuffer++;
            }
            if (found) {
                /* se achou, ignora todo o conteudo anterior
                   ao delimitador */
                offset = idxBuffer - begin.length;
                len = buffer.length - offset;
                break;
            }
            idxBuffer++;
        }
        /* se nao achou delimitador, considera o conteudo na
           integra, pode estar em formato binario (DER) */
        ByteArrayInputStream internalStream =
            new ByteArrayInputStream(buffer,offset,len);
        /* retorna um InputStream baseado no buffer interno */
        return internalStream;
    }
}
```

com.robsonmartins.fiap.tcc.util.CertificadoAssinador.java

```
package com.robsonmartins.fiap.tcc.util;

import java.io.FileInputStream;
import java.io.InputStream;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.PublicKey;
import java.security.Signature;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;

/**
 * Implementa metodos para assinar conteudos com certificados digitais.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class CertificadoAssinador {
    /**
     * Retorna um par de chaves armazenado em um arquivo de keystore.
     * @param keyStoreFile Nome do arquivo de keystore.
     * @param keyStoreType Tipo do arquivo de keystore.
     * @param keyAlias Alias do par de chaves dentro do keystore.
     * @param keyStorePass Senha do keystore.
     * @param keyPass Senha do par de chaves dentro do keystore.
     * @return Par de chaves (publica e privada).
     * @throws Exception
     */
    public static KeyPair getKeyPairFromFile(String keyStoreFile,
            String keyStoreType, String keyAlias,
            String keyStorePass, String keyPass) throws Exception {

        PrivateKey privateKey = null;
        PublicKey  publicKey  = null;
        KeyPair    keys       = null;
        KeyStore keyStore =
            loadKeyStoreFromFile(keyStoreFile, keyStorePass, keyStoreType);
        Key key = keyStore.getKey(keyAlias, keyPass.toCharArray());
        if (key instanceof PrivateKey) {
            privateKey = (PrivateKey) key;
        }
        Certificate cert = keyStore.getCertificate(keyAlias);
        publicKey = cert.getPublicKey();
        if (publicKey != null && privateKey != null) {
            keys = new KeyPair(publicKey, privateKey);
        }
```

com.robsonmartins.fiap.tcc.util.CertificadoAssinador.java

```
        return keys;
    }

    /**
     * Retorna um par de chaves armazenado em um arquivo de keystore (JKS).
     * @param keyStoreFile Nome do arquivo de keystore (JKS).
     * @param keyAlias Alias do par de chaves dentro do keystore.
     * @param keyStorePass Senha do keystore.
     * @param keyPass Senha do par de chaves dentro do keystore.
     * @return Par de chaves (publica e privada).
     * @throws Exception
     */
    public static KeyPair getKeyPairFromFile(String keyStoreFile,
          String keyAlias, String keyStorePass, String keyPass) throws Exception {
        return getKeyPairFromFile(
            keyStoreFile, "JKS", keyAlias, keyStorePass, keyPass);
    }

    /**
     * Retorna um certificado armazenado em um arquivo de keystore.
     * @param keyStoreFile Nome do arquivo de keystore.
     * @param keyStoreType Tipo do arquivo de keystore.
     * @param keyAlias Alias do par de chaves dentro do keystore.
     * @param keyStorePass Senha do keystore.
     * @return Objeto que representa um certificado digital.
     * @throws Exception
     */
    public static X509Certificate getCertFromFile(String keyStoreFile,
            String keyStoreType, String keyAlias,
            String keyStorePass) throws Exception {

        KeyStore keyStore =
            loadKeyStoreFromFile(keyStoreFile, keyStorePass, keyStoreType);
        return (X509Certificate) keyStore.getCertificate(keyAlias);
    }

    /**
     * Retorna um certificado armazenado em um arquivo de keystore (JKS).
     * @param keyStoreFile Nome do arquivo de keystore (JSK).
     * @param keyAlias Alias do par de chaves dentro do keystore.
     * @param keyStorePass Senha do keystore.
     * @return Objeto que representa um certificado digital.
     * @throws Exception
     */
    public static X509Certificate getCertFromFile(String keyStoreFile,
            String keyAlias, String keyStorePass) throws Exception {
```

com.robsonmartins.fiap.tcc.util.CertificadoAssinador.java

```
        return getCertFromFile(keyStoreFile, "JKS", keyAlias, keyStorePass);
    }

    /**
     * Assina um conteudo usando uma chave privada.
     * @param provider Provider de criptografia.
     * @param algorithm Algoritmo de assinatura a ser usado.
     * @param key Chave privada.
     * @param content Conteudo a ser assinado.
     * @return Assinatura do conteudo.
     * @throws Exception
     */
    public static byte[] sign(Provider provider, String algorithm, PrivateKey key,
            byte[] content) throws Exception {
        if (algorithm == null) { algorithm = "SHA1withRSA"; }
        if (provider  == null) { return sign(algorithm, key, content); }
        Signature sig = Signature.getInstance(algorithm, provider);
        sig.initSign(key);
        sig.update(content, 0, content.length);
        return sig.sign();
    }

    /**
     * Assina um conteudo usando uma chave privada
     *   (provider default).
     * @param algorithm Algoritmo de assinatura a ser usado.
     * @param key Chave privada.
     * @param content Conteudo a ser assinado.
     * @return Assinatura do conteudo.
     * @throws Exception
     */
    public static byte[] sign(String algorithm, PrivateKey key,
            byte[] content) throws Exception {

        Signature sig = Signature.getInstance(algorithm);
        sig.initSign(key);
        sig.update(content, 0, content.length);
        return sig.sign();
    }
```

com.robsonmartins.fiap.tcc.util.CertificadoAssinador.java

```
/**
 * Assina um conteudo usando uma chave privada
 *   (provider default, algoritmo SHA1 com RSA).
 * @param key Chave privada.
 * @param content Conteudo a ser assinado.
 * @return Assinatura do conteudo.
 * @throws Exception
 */
public static byte[] sign(PrivateKey key,
        byte[] content) throws Exception {

    return sign("SHA1withRSA", key, content);
}

/**
 * Verifica a autenticidade de um conteudo.
 * @param provider Provider de criptografia.
 * @param algorithm Algoritmo de assinatura a ser usado.
 * @param key Chave publica.
 * @param content Conteudo a ser verificado.
 * @param signed Assinatura do conteudo.
 * @return True se autenticidade foi confirmada, false se nao.
 * @throws Exception
 */
public static boolean verify(Provider provider, String algorithm,
        PublicKey key, byte[] content, byte[] signed) throws Exception {

    if (algorithm == null) { algorithm = "SHA1withRSA"; }
    if (provider  == null) { return verify(algorithm, key, content, signed); }
    Signature sig = Signature.getInstance(algorithm, provider);
    sig.initVerify(key);
    sig.update(content, 0, content.length);
    return sig.verify(signed);
}

/**
 * Verifica a autenticidade de um conteudo
 *   (provider default).
 * @param algorithm Algoritmo de assinatura a ser usado.
 * @param key Chave publica.
 * @param content Conteudo a ser verificado.
 * @param signed Assinatura do conteudo.
 * @return True se autenticidade foi confirmada, false se nao.
 * @throws Exception
 */
public static boolean verify(String algorithm, PublicKey key,
        byte[] content, byte[] signed) throws Exception {
```

com.robsonmartins.fiap.tcc.util.CertificadoAssinador.java

```
        Signature sig = Signature.getInstance(algorithm);
        sig.initVerify(key);
        sig.update(content, 0, content.length);
        return sig.verify(signed);
    }

    /**
     * Verifica a autenticidade de um conteudo
     *   (provider default, algoritmo SHA1 com RSA).
     * @param key Chave publica.
     * @param content Conteudo a ser verificado.
     * @param signed Assinatura do conteudo.
     * @return True se autenticidade foi confirmada, false se nao.
     * @throws Exception
     */
    public static boolean verify(PublicKey key, byte[] content,
            byte[] signed) throws Exception {

        return verify("SHA1withRSA", key, content, signed);
    }

    /* Retorna um objeto {@link KeyStore} a partir de um arquivo.
     * @param keyStoreFile Nome do arquivo de keystore.
     * @param keyStorePass Senha do keystore.
     * @param keyStoreType Tipo do arquivo de keystore.
     * @return Objeto KeyStore.
     * @throws Exception
     */
    private static KeyStore loadKeyStoreFromFile(String keyStoreFile,
            String keyStorePass, String keyStoreType) throws Exception {

        KeyStore keyStore = KeyStore.getInstance(keyStoreType);
        InputStream istream = new FileInputStream(keyStoreFile);
        keyStore.load(istream, keyStorePass.toCharArray());
        return keyStore;
    }
}
```

4.2 Módulo de *Login* JAAS para o SICid

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
package com.robsonmartins.fiap.tcc.sicid.jaas;

import java.security.Principal;
import java.security.acl.Group;
import java.security.cert.X509Certificate;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Arrays;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.LoginException;
import javax.sql.DataSource;
import javax.transaction.SystemException;
import javax.transaction.Transaction;
import javax.transaction.TransactionManager;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.jboss.security.NestableGroup;
import org.jboss.security.SimpleGroup;
import org.jboss.security.auth.spi.DatabaseServerLoginModule;
import org.jboss.security.plugins.TransactionManagerLocator;
import sun.misc.BASE64Decoder;

import com.robsonmartins.fiap.tcc.util.CertificadoAssinador;
import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;
import sicid.ws.SICidClient;
import sicid.bean.CertificadoStatus;

/**
 * LoginModule para uso com o JAAS, baseado em banco de dados (JDBC).<br/>
 * Implementa a validacao de certificados digitais atraves de consulta ao
 *   Servico de Identificacao do Cidadao (SICid).
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
 *
 * <p>
 *  Tabelas:
 * <ul>
 * <li>Principals(DN text, PrincipalID text)
 * <li>Roles(PrincipalID text, Role text, RoleGroup text)
 * </ul>
* <p>
 * Opcoes do LoginModule:
 * <ul>
 * <li><em>dsJndiName</em>: O nome do DataSource do banco de dados que contem
 *   as tabelas Principals e Roles.
 * <li><em>principalsQuery</em>: A consulta SQL para obter um nome de usuario
 *   mapeado a partir de um Distinguished Name (DN) do certificado digital,
 * equivalente a:
 * <pre>
 *    "select PrincipalID from Principals where DN=?"
 * </pre>
 * <li><em>rolesQuery</em>: A consulta SQL para obter os Roles e RoleGroups
 *   a partir de um nome de usuario, equivalente a:
 * <pre>
 *    "select Role, RoleGroup from Roles where PrincipalID=?"
 * </pre>
 * <li><em>fullPrincipalsQuery</em>: A consulta SQL para obter todos usuarios
 * cadastrados, equivalente a:
 * <pre>
 *    "select * from Principals"
 * </pre>
 * <li><em>principalIdForEmpty</em>: Id a ser atribuida a um usuario caso nao haja
 *   usuarios cadastrados.
 * <li><em>roleForEmpty</em>: Role a ser atribuida a um usuario
 * caso nao haja usuarios cadastrados.
 * <li><em>sicid.disable</em>: Desabilita a consulta ao Servico de Identificacao
 *   do Cidadao (SICid), considerando qualquer certificado como valido.
 * <li><em>sicid.clientProps</em>: Nome do arquivo que contem a configuracao do
 *   cliente do Servico de Identificacao do Cidadao (SICid). Se especificado,
 *   todas as opcoes seguintes sao ignoradas, pois sao obtidas a partir desse
 *   arquivo.
 * <li><em>sicid.url</em>: URL do Servico de Identificacao do Cidadao (SICid).
 * <li><em>sicid.namespace</em>: Namespace do Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.service</em>: Nome do Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.keyStore</em>: Nome do arquivo de keystore contendo o par de
 *   chaves do cliente para autenticar no Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.storeType</em>: Tipo do arquivo de keystore (default: "JKS").
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
 * <li><em>sicid.storePass</em>: Senha do arquivo de keystore.
 * <li><em>sicid.keyAlias</em>: Alias do par de chaves a ser usado, dentro do
 *   arquivo de keystore.
 * <li><em>sicid.keyPass</em>: Senha do par de chaves dentro do arquivo de
 *   keystore.
 * </ul>
 * <p>
 *
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class SICidDBLoginModule extends DatabaseServerLoginModule {
    /* nomes das opcoes do loginModule no arquivo login-config.xml */
    private final static String OPTION_FULL_QUERY         = "fullPrincipalsQuery";
    private final static String OPTION_ID_FOR_EMPTY       = "principalIdForEmpty";
    private final static String OPTION_ROLE_FOR_EMPTY     = "roleForEmpty"       ;
    private final static String OPTION_SICID_CLIENT_PROPS = "sicid.clientProps"  ;
    private final static String OPTION_SICID_DISABLE      = "sicid.disable"      ;
    private final static String OPTION_SICID_URL          = "sicid.url"          ;
    private final static String OPTION_SICID_NAMESPACE    = "sicid.namespace"    ;
    private final static String OPTION_SICID_SERVICE      = "sicid.service"      ;
    private final static String OPTION_SICID_KEYSTORE     = "sicid.keyStore"     ;
    private final static String OPTION_SICID_KEYSTORE_TYPE = "sicid.storeType"  ;
    private final static String OPTION_SICID_KEYSTORE_PASS = "sicid.storePass"  ;
    private final static String OPTION_SICID_KEY_ALIAS     = "sicid.keyAlias"   ;
    private final static String OPTION_SICID_KEY_PASS      = "sicid.keyPass"    ;
    private Subject subject;
    private CallbackHandler callbackHandler;
    private static Logger log;
    private static boolean trace;
    private String dname;
    private Principal identity;
    private Group[] roles;
    private String fullPrincipalsQuery;
    private String principalIdForEmpty;
    private String roleForEmpty;
    private String sicidDisable;
    private String sicidClientProps;
    private String sicidServiceURL;
    private String sicidNamespace;
    private String sicidService;
    private String sicidKeyStoreFile;
    private String sicidKeyStoreType;
    private String sicidKeyStorePass;
    private String sicidKeyAlias;
    private String sicidKeyPass;
    private boolean isPrincipalsEmpty;
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
/* cliente do servico SICid */
private SICidClient sicid = null;
/* indica se a consulta ao servico SICid esta' habilitada */
private boolean sicidEnabled = true;

/**
 * Initialize this LoginModule.
 * @param subject the Subject to update after a successful login.
 * @param callbackHandler the CallbackHandler that will be used to obtain the
 *    the user identity and credentials.
 * @param sharedState a Map shared between all configured
 *    login module instances
 * @param options the parameters passed to the login module.
 */
@Override
public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map<String, ?> sharedState, Map<String, ?> options) {

    log = LogManager.getLogger(SICidDBLoginModule.class);
    trace = log.isTraceEnabled();
    sicid = new SICidClient();
    super.initialize(subject, callbackHandler, sharedState, options);
    Map<String, String> loginModuleOpts = parseOptions(options);
    fullPrincipalsQuery = loginModuleOpts.get(OPTION_FULL_QUERY         );
    principalIdForEmpty = loginModuleOpts.get(OPTION_ID_FOR_EMPTY       );
    roleForEmpty        = loginModuleOpts.get(OPTION_ROLE_FOR_EMPTY     );
    sicidDisable        = loginModuleOpts.get(OPTION_SICID_DISABLE      );
    sicidClientProps    = loginModuleOpts.get(OPTION_SICID_CLIENT_PROPS );
    sicidServiceURL     = loginModuleOpts.get(OPTION_SICID_URL          );
    sicidNamespace      = loginModuleOpts.get(OPTION_SICID_NAMESPACE    );
    sicidService        = loginModuleOpts.get(OPTION_SICID_SERVICE      );
    sicidKeyStoreFile   = loginModuleOpts.get(OPTION_SICID_KEYSTORE     );
    sicidKeyStoreType   = loginModuleOpts.get(OPTION_SICID_KEYSTORE_TYPE);
    sicidKeyStorePass   = loginModuleOpts.get(OPTION_SICID_KEYSTORE_PASS);
    sicidKeyAlias       = loginModuleOpts.get(OPTION_SICID_KEY_ALIAS    );
    sicidKeyPass        = loginModuleOpts.get(OPTION_SICID_KEY_PASS     );
    this.subject = subject;
    this.callbackHandler = callbackHandler;
    sicidEnabled = !("true".equalsIgnoreCase(sicidDisable));
    isPrincipalsEmpty = false;
    if (sicidKeyStorePass == null) { sicidKeyStorePass = ""; }
    if (sicidKeyPass      == null) { sicidKeyPass      = ""; }
    if (sicidKeyAlias     == null) { sicidKeyAlias     = ""; }
    if (sicidKeyStoreType == null || "".equals(sicidKeyStoreType)) {
        sicidKeyStoreType = "JKS";
    }
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
        if (trace) {
            log.trace(String.format("%s=%s", OPTION_FULL_QUERY         ,
                                    fullPrincipalsQuery));
            log.trace(String.format("%s=%s", OPTION_ID_FOR_EMPTY       ,
                                    principalIdForEmpty));
            log.trace(String.format("%s=%s", OPTION_ROLE_FOR_EMPTY     ,
                                    roleForEmpty       ));
            log.trace(String.format("%s=%s", OPTION_SICID_DISABLE      ,
                                    sicidDisable       ));
            log.trace(String.format("%s=%s", OPTION_SICID_CLIENT_PROPS ,
                                    sicidClientProps   ));
            log.trace(String.format("%s=%s", OPTION_SICID_URL          ,
                                    sicidServiceURL    ));
            log.trace(String.format("%s=%s", OPTION_SICID_NAMESPACE    ,
                                    sicidNamespace     ));
            log.trace(String.format("%s=%s", OPTION_SICID_SERVICE      ,
                                    sicidService       ));
            log.trace(String.format("%s=%s", OPTION_SICID_KEYSTORE     ,
                                    sicidKeyStoreFile  ));
            log.trace(String.format("%s=%s", OPTION_SICID_KEYSTORE_TYPE,
                                    sicidKeyStoreType  ));
            log.trace(String.format("%s=%s", OPTION_SICID_KEYSTORE_PASS,
                                    sicidKeyStorePass  ));
            log.trace(String.format("%s=%s", OPTION_SICID_KEY_ALIAS    ,
                                    sicidKeyAlias      ));
            log.trace(String.format("%s=%s", OPTION_SICID_KEY_PASS     ,
                                    sicidKeyPass       ));
        }
    }

    /**
     * Perform the authentication of the username and password.
     * @return True if success.
     */
    @Override
    public boolean login() throws LoginException {
        try {
            super.loginOk = false;
            NameCallback nameCallback =
                new NameCallback("Username: ");
            PasswordCallback passwordCallback =
                new PasswordCallback("Password: ", false);
            Callback[] callbacks = {nameCallback, passwordCallback};
            callbackHandler.handle(callbacks);
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
/* obtem username e password atraves do JAAS */
String name =
    new String(nameCallback.getName())
        .replaceAll("\\s+","");
String pass =
    new String(passwordCallback.getPassword())
        .replaceAll("\\s+","");
passwordCallback.clearPassword();
/* username contem um certificado codificado em Base64
   e password contem a assinatura do conteudo do certificado,
   tambem codificada em Base64 */
if (trace) {
    log.trace(String.format("Cert content: %s",name));
    log.trace(String.format("Sign content: %s",pass));
}
/* desserializa o certificado */
X509Certificate cert = null;
try {
    cert = CertificadoSerializador.strToCert(name);
} catch (Exception e) {
    LoginException exception =
        new LoginException("Error getting the certificate");
    exception.initCause(e);
    throw exception;
}
isPrincipalsEmpty = isPrincipalsEmpty();
/* obtem o DN do certificado */
dname = cert.getSubjectX500Principal().getName();
/* obtem o username (via DB query) associado ao DN */
String username = (String) getUsersPassword();
if (trace) {
    log.trace(String.format("Login: certificate DN: %s", dname));
}
/* executa procedimentos de validacao do certificado */
try {
    validateCert(name, pass);
} catch (Exception e) {
    LoginException exception =
        new LoginException(String.format(
                "Invalid Certificate %s", dname));
    exception.initCause(e);
    throw exception;
}
if (username == null) {
    LoginException exception =
        new LoginException(String.format(
                "No user mapping for DN: %s", dname));
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
                throw exception;
            }
            if (trace) {
                log.trace(String.format("Username: %s", username));
            }
            /* cria objeto identity que representa o usuario autenticado */
            if (this.identity == null) {
                try {
                    this.identity = createIdentity(username);
                } catch (Exception e) {
                    if (trace) {
                        log.trace("Failed to create principal", e);
                    }
                    LoginException exception =
                        new LoginException("Failed to create principal");
                    exception.initCause(e);
                    throw exception;
                }
            }
            /* obtem roles do usuario autenticado */
            this.roles = getRoleSets();
            if (trace) {
                log.trace(String.format("Roles: %s", Arrays.toString(roles)));
            }
            super.loginOk = true;
            if (trace) {
                log.trace("User '" + this.identity
                          + "' authenticated, loginOk=" + loginOk);
            }
            return true;
        } catch (Exception e) {
            log.error("Login error", e);
            LoginException exception =
                new LoginException("Login error");
            exception.initCause(e);
            throw exception;
        }
    }

    /**
     * Method to commit the authentication process (phase 2). If the login
     * method completed successfully as indicated by loginOk == true, this
     * method adds the getIdentity() value to the subject getPrincipals() Set.
     * It also adds the members of each Group returned by getRoleSets()
     * to the subject getPrincipals() Set.
     * @return true always.
     */
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
@Override
public boolean commit() throws LoginException {
    if (identity != null) {
        Set<Principal> principals = subject.getPrincipals();
        principals.add(identity);
        if (roles != null) {
            for (Group group : roles) {
                String name = group.getName();
                Group subjectGroup = createGroup(name, principals);
                if (subjectGroup instanceof NestableGroup) {
                    SimpleGroup sg = new SimpleGroup("Roles");
                    subjectGroup.addMember(sg);
                    subjectGroup = sg;
                }
                Enumeration<? extends Principal> members = group.members();
                while (members.hasMoreElements()) {
                    Principal role = (Principal) members.nextElement();
                    subjectGroup.addMember(role);
                }
            }
        }
    }
    return true;
}

/**
 * Method to abort the authentication process (phase 2).
 * @return true alaways.
 */
@Override
public boolean abort() throws LoginException {
    this.subject         = null;
    this.callbackHandler = null;
    this.identity        = null;
    return true;
}

/**
 * Remove the user identity and roles added to the Subject during commit.
 * @return true always.
 */
@Override
public boolean logout() throws LoginException {
    if (identity != null) {
        Set<Principal> principals = subject.getPrincipals();
        principals.remove(identity);
    }
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
        return true;
    }

    /**
     * Get the expected password for the current username available via
     * the getUsername() method. This is called from within the login()
     * method after the CallbackHandler has returned the username and
     * candidate password.
     * @return the valid password String.
     */
    @Override
    protected String getUsersPassword() throws LoginException {
        /* obtem username associado ao DN do certificado, via DB query */
        if (isPrincipalsEmpty) {
            return principalIdForEmpty;
        }
        String username = null;
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        Transaction tx = null;
        if (suspendResume) {
            try {
                if (tm == null) {
                    throw new
                         IllegalStateException("Transaction Manager is null");
                }
                tx = tm.suspend();
            } catch (SystemException e)    {
                throw new RuntimeException(e);
            }
            if (trace) {
                log.trace("suspendAnyTransaction");
            }
        }
        try {
            InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup(dsJndiName);
            conn = ds.getConnection();
            if (trace) {
                log.trace(
                        String.format("Executing query: %s, with DN: %s",
                                principalsQuery, dname));
            }
            ps = conn.prepareStatement(principalsQuery);
            ps.setString(1, dname);
            rs = ps.executeQuery();
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            if (rs.next() == false) {
                if (trace) {
                    log.trace("Query returned no matches from db");
                }
                throw new FailedLoginException(
                        "No matching username found in Principals");
            }
            username = rs.getString(1);
            if (trace) {
                log.trace("Obtained username");
            }
        } catch (NamingException ex) {
            LoginException le =
                new LoginException(
                        String.format(
                            "Error looking up DataSource from: %s", dsJndiName));
            le.initCause(ex);
            throw le;
        } catch(SQLException ex) {
            LoginException le = new LoginException("Query failed");
            le.initCause(ex);
            throw le;
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {}
            }
            if (ps != null) {
                try {
                    ps.close();
                } catch(SQLException e) {}
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException ex) {}
            }
            if (suspendResume) {
                try {
                    tm.resume(tx);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
                if (trace) {
                    log.trace("resumeAnyTransaction");
                }
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            }
        }
        return username;
    }

    /**
     * Returns the username.
     * @return the username.
     */
    @Override
    protected String getUsername() {
        String username = null;
        if (getIdentity() != null) {
            username = getIdentity().getName();
        }
        return username;
    }

    /**
     * Returns the Principal that corresponds to
     * the user primary identity.
     * @return the user primary identity.
     */
    @Override
    protected Principal getIdentity() {
        return this.identity;
    }

    /** Execute the rolesQuery against the dsJndiName to obtain the roles for
     *  the authenticated user.
     *  @return Group[] containing the sets of roles
     */
    @Override
    protected Group[] getRoleSets() throws LoginException {
        Group[] roleSets = null;
        if (isPrincipalsEmpty) {
            roleSets = new Group[1];
            SimpleGroup sg = new SimpleGroup("Roles");
            try {
                Principal p = createIdentity(roleForEmpty);
                sg.addMember(p);
            } catch (Exception e) {
                if (trace) {
                    log.trace(String.format(
                        "Failed to create principal: %s", roleForEmpty), e);
                }
            }
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            roleSets[0] = sg;
            return roleSets;
        }
        String username = getUsername();
        if (trace) {
            log.trace(String.format(
                    "getRoleSets using rolesQuery: %s, username: %s",
                    rolesQuery, username));
        }
        roleSets =
            getRoleSets(username, dsJndiName, rolesQuery, suspendResume);
        return roleSets;
    }

    /* Retorna as opcoes do LoginModule.
     * @param options Opcoes passadas via arquivo login-config.xml.
     * @return Opcoes do LoginModule.
     */
    private Map<String, String> parseOptions(Map<String, ?> options) {
        Map<String, String> optList = new HashMap<String, String>();
        String fullQuery    = (String) options.get(OPTION_FULL_QUERY         );
        String idForEmpty   = (String) options.get(OPTION_ID_FOR_EMPTY       );
        String roleForEmpty = (String) options.get(OPTION_ROLE_FOR_EMPTY     );
        String disable      = (String) options.get(OPTION_SICID_DISABLE      );
        String clientProps  = (String) options.get(OPTION_SICID_CLIENT_PROPS );
        String serviceURL   = (String) options.get(OPTION_SICID_URL          );
        String namespace    = (String) options.get(OPTION_SICID_NAMESPACE    );
        String service      = (String) options.get(OPTION_SICID_SERVICE      );
        String keyStoreFile = (String) options.get(OPTION_SICID_KEYSTORE     );
        String keyStoreType = (String) options.get(OPTION_SICID_KEYSTORE_TYPE);
        String keyStorePass = (String) options.get(OPTION_SICID_KEYSTORE_PASS);
        String keyAlias     = (String) options.get(OPTION_SICID_KEY_ALIAS    );
        String keyPass      = (String) options.get(OPTION_SICID_KEY_PASS     );
        if (clientProps != null && !"".equals(clientProps)) {
            try {
                Properties props = sicid.getOptionsFromFile(clientProps);
                serviceURL   = props.getProperty(OPTION_SICID_URL          );
                namespace    = props.getProperty(OPTION_SICID_NAMESPACE    );
                service      = props.getProperty(OPTION_SICID_SERVICE      );
                keyStoreFile = props.getProperty(OPTION_SICID_KEYSTORE     );
                keyStoreType = props.getProperty(OPTION_SICID_KEYSTORE_TYPE);
                keyStorePass = props.getProperty(OPTION_SICID_KEYSTORE_PASS);
                keyAlias     = props.getProperty(OPTION_SICID_KEY_ALIAS    );
                keyPass      = props.getProperty(OPTION_SICID_KEY_PASS     );
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            } catch (Exception e) {
                if (trace) {
                    log.error("Error getting SICid Client options", e);
                }
            }
        }
        optList.put(OPTION_FULL_QUERY         , fullQuery   );
        optList.put(OPTION_ID_FOR_EMPTY       , idForEmpty  );
        optList.put(OPTION_ROLE_FOR_EMPTY     , roleForEmpty);
        optList.put(OPTION_SICID_DISABLE      , disable     );
        optList.put(OPTION_SICID_CLIENT_PROPS , clientProps );
        optList.put(OPTION_SICID_URL          , serviceURL  );
        optList.put(OPTION_SICID_NAMESPACE    , namespace   );
        optList.put(OPTION_SICID_SERVICE      , service     );
        optList.put(OPTION_SICID_KEYSTORE     , keyStoreFile);
        optList.put(OPTION_SICID_KEYSTORE_TYPE, keyStoreType);
        optList.put(OPTION_SICID_KEYSTORE_PASS, keyStorePass);
        optList.put(OPTION_SICID_KEY_ALIAS    , keyAlias    );
        optList.put(OPTION_SICID_KEY_PASS     , keyPass     );
        return optList;
    }

    /* Valida um certificado, realizando uma consulta ao servico SICid.
     * @param content Conteudo do certificado, codificado em Base64.
     * @param signed Assinatura do certificado, codificada em Base64.
     * @throws Exception
     */
    private void validateCert(String content, String signed) throws Exception {
        CertificadoStatus status = CertificadoStatus.INVALID;
        try {
            /* desserializa certificado */
            X509Certificate cert = CertificadoSerializador.strToCert(content);
            /* decodifica assinatura para buffer */
            byte[] sigBuffer = new BASE64Decoder().decodeBuffer(signed);
            if (trace) {
                log.trace(String.format("Validating certificate: %s",
                        cert.getSubjectX500Principal().getName()));
            }
            /* vertifica assinatura do certificado */
            if (!CertificadoAssinador.verify(cert.getPublicKey(),
                    cert.getEncoded(), sigBuffer)) {
                throw new Exception("Wrong signature");
            }
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            /* se parametro sicid.disable!=true do login-config.xml,
             * faz consulta ao SICid */
            if (sicidEnabled) {
                try {
                    /* valida certificado atraves do servico SICid */
                    sicid.connect(sicidServiceURL, sicidNamespace, sicidService,
                            sicidKeyStoreFile, sicidKeyStoreType, sicidKeyAlias,
                            sicidKeyStorePass, sicidKeyPass);
                    status = sicid.getService().validarCertificado(content);
                } catch (Exception e) {
                    log.error("Failed to connect in SICid service", e);
                    Exception exception =
                        new Exception("Failed to connect in SICid service");
                    exception.initCause(e);
                    throw exception;
                }
            } else {
                status = CertificadoStatus.VALID;
            }
            if (status != CertificadoStatus.VALID) {
                throw new Exception(
                        String.format("Certificate status: %s", status));
            }
            if (trace) {
                log.trace(String.format("Certificate status: %s", status));
            }
        } catch (Exception e) {
            if (trace) {
                log.trace(String.format("Certificate: %s", status));
            }
            Exception exception = new Exception("Invalid certificate");
            exception.initCause(e);
            throw exception;
        }
    }

    /* Retorna true se a tabela de usuarios esta' vazia.
     * @return True se nao ha' usuarios cadastrados.
     * @throws Exception
     */
    private boolean isPrincipalsEmpty() {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        TransactionManager tm = null;
        if (suspendResume) {
            TransactionManagerLocator tml = new TransactionManagerLocator();
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            try {
                tm = tml.getTM("java:/TransactionManager");
            } catch (NamingException e1) {
                if (trace) {
                    log.trace("Error in Transaction Manager", e1);
                }
                return false;
            }
            if (tm == null) {
                if (trace) {
                    log.trace("Transaction Manager is null");
                }
                return false;
            }
        }
        Transaction tx = null;
        if (suspendResume) {
            try {
                tx = tm.suspend();
            } catch (SystemException e)    {
                if (trace) {
                    log.trace("Error suspending transaction", e);
                }
                return false;
            }
            if (trace) {
                log.trace("suspendAnyTransaction");
            }
        }
        try {
            InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup(dsJndiName);
            conn = ds.getConnection();
            if (trace) {
                log.trace(String.format(
                        "Executing query: %s", fullPrincipalsQuery));
            }
            ps = conn.prepareStatement(fullPrincipalsQuery);
            rs = ps.executeQuery();
            if (rs.next() == false) {
                if (trace) {
                    log.trace("Principals table is empty");
                }
                return true;
            }
        } catch (Exception e) {
            if (trace) {
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
                log.trace("Error executing query", e);
            }
            return false;
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {}
            }
            if (ps != null) {
                try {
                    ps.close();
                } catch(SQLException e)    {}
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (Exception ex) {}
            }
            if (suspendResume) {
                try {
                    tm.resume(tx);
                } catch (Exception e) {
                    if (trace) {
                        log.trace("Error resuming transaction", e);
                    }
                    return false;
                }
                if (trace) {
                    log.trace("resumeAnyTransaction");
                }
            }
        }
        if (trace) {
            log.trace("Principals table is not empty");
        }
        return false;
    }


    /*
     * Execute the rolesQuery against the dsJndiName to obtain the roles
     *   for the authenticated user.
     * @return Group[] containing the sets of roles
     *
     * Source: org.jboss.security.auth.spi.DbUtil.java
     */
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
private Group[] getRoleSets(String username, String dsJndiName,
        String rolesQuery, boolean suspendResume) throws LoginException {

    /* obtem roles de um usuario, a partir de uma DB query */
    Connection conn = null;
    HashMap<String,Group> setsMap = new HashMap<String,Group>();
    PreparedStatement ps = null;
    ResultSet rs = null;
    TransactionManager tm = null;
    if (suspendResume) {
        TransactionManagerLocator tml = new TransactionManagerLocator();
        try {
            tm = tml.getTM("java:/TransactionManager");
        } catch (NamingException e1) {
            throw new RuntimeException(e1);
        }
        if (tm == null) {
            throw new IllegalStateException("Transaction Manager is null");
        }
    }
    Transaction tx = null;
    if (suspendResume) {
        try {
            tx = tm.suspend();
        } catch (SystemException e) {
            throw new RuntimeException(e);
        }
        if (trace) {
            log.trace("suspendAnyTransaction");
        }
    }
    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsJndiName);
        conn = ds.getConnection();
        if (trace) {
            log.trace(String.format(
                    "Executing query: %s, with username: %s",
                    rolesQuery, username));
        }
        ps = conn.prepareStatement(rolesQuery);
        try {
            ps.setString(1, username);
        } catch (ArrayIndexOutOfBoundsException ignore) {}
        rs = ps.executeQuery();
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
            if (rs.next() == false) {
                if (trace) {
                    log.trace("No roles found");
                }
                if (getUnauthenticatedIdentity() == null) {
                    throw new FailedLoginException(
                        "No matching username found in Roles");
                }
                Group[] roleSets = { new SimpleGroup("Roles") };
                return roleSets;
            }
            do {
                String name = rs.getString(1);
                String groupName = rs.getString(2);
                if (groupName == null || groupName.length() == 0) {
                    groupName = "Roles";
                }
                Group group = (Group) setsMap.get(groupName);
                if (group == null) {
                    group = new SimpleGroup(groupName);
                    setsMap.put(groupName, group);
                }
                try {
                    Principal p = createIdentity(name);
                    if (trace) {
                        log.trace(String.format(
                                "Assign user to role %s", name));
                    }
                    group.addMember(p);
                }
                catch (Exception e) {
                    if (trace) {
                        log.trace(String.format(
                            "Failed to create principal: %s", name), e);
                    }
                }
            } while (rs.next());
        } catch (NamingException ex) {
            LoginException le =
                new LoginException(String.format(
                        "Error looking up DataSource from: %s", dsJndiName));
            le.initCause(ex);
            throw le;
        } catch (SQLException ex) {
            LoginException le = new LoginException("Query failed");
            le.initCause(ex);
            throw le;
```

com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule.java

```
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {}
            }
            if (ps != null) {
                try {
                    ps.close();
                } catch(SQLException e)    {}
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (Exception ex) {}
            }
            if (suspendResume) {
                try {
                    tm.resume(tx);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
                if (trace) {
                    log.trace("resumeAnyTransaction");
                }
            }
        }
        Group[] roleSets = new Group[setsMap.size()];
        setsMap.values().toArray(roleSets);
        return roleSets;
    }
}
```

4.3 Serviço (*Web Service*) do SICid

sicid.ws.ISICidService.java

```
package sicid.ws;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

import sicid.bean.CertificadoStatus;
import sicid.bean.Cidadao;
import sicid.bean.CidadaoCollecion;

/**
 * Interface do Servico de Identificacao do Cidadao (SICid).
 * @author Robson Martins (robson@robsonmartins.com)
 */
@WebService(name="SICidService")
@SOAPBinding(style=Style.DOCUMENT)
public interface ISICidService {
    /**
     * Valida um certificado digital.
     * @param content Conteudo de um certificado, codificado em Base64.
     * @return Status do certificado.
     */
    @WebMethod
    @WebResult(name="status")
    public CertificadoStatus validarCertificado(
            @WebParam(name="certificado") String content);

    /**
     * Consulta os dados cadastrais de um cidadao,
     *   a partir de seu certificado digital.
     * @param content Conteudo de um certificado, codificado em Base64.
     * @return Dados cadastrais de um cidadao.
     */
    @WebMethod
    @WebResult(name="cidadao")
    public Cidadao consultarCidadao(
            @WebParam(name="certificado") String content);
```

sicid.ws.ISICidService.java

```
    /**
     * Retorna uma lista de todos cidadaos cadastrados.
     * @return Lista de cidadaos cadastrados.
     */
    @WebMethod
    @WebResult(name="cidadaos")
    public CidadaoCollecion listarCidadaos();
}
```

sicid.ws.SICidService.java

```
package sicid.ws;

import javax.annotation.security.RolesAllowed;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.jws.WebService;
import org.jboss.security.annotation.SecurityDomain;
import org.jboss.wsf.spi.annotation.WebContext;

import sicid.bean.CertificadoStatus;
import sicid.bean.Cidadao;
import sicid.bean.CidadaoCollecion;
import sicid.model.ISICidEngine;

/**
 * Implementacao do Servico de Identificacao do Cidadao (SICid).
 * @author Robson Martins (robson@robsonmartins.com)
 */
@WebService(serviceName = "SICidService", portName = "SICidPort",
    targetNamespace = "http://ws.sicid/",
    endpointInterface = "sicid.ws.ISICidService")
@SecurityDomain("SICidService")
@RolesAllowed("wsuser")
@WebContext(contextRoot = "/sicid/service", urlPattern = "/sicid",
    transportGuarantee = "CONFIDENTIAL", authMethod = "BASIC",
    secureWSDLAccess = true)
@Stateless
public class SICidService implements ISICidService {
    /* Instancia do motor (MVC 'model') do SICid */
    @EJB
    private ISICidEngine sicidEngine;

    @Override
    public CertificadoStatus validarCertificado(String content) {
        return sicidEngine.validarCertificado(content);
    }

    @Override
    public Cidadao consultarCidadao(String content) {
        return sicidEngine.consultarCidadao(content);
    }
```

sicid.ws.SICidService.java

```
    @Override
    public CidadaoCollecion listarCidadaos() {
        CidadaoCollecion cidadaos = new CidadaoCollecion();
        cidadaos.setCidadaos(sicidEngine.listarCidadaos());
        return cidadaos;
    }
}
```

sicid.bean.CertificadoStatus.java

```
package sicid.bean;

import javax.xml.bind.annotation.XmlEnum;
import javax.xml.bind.annotation.XmlType;

/**
 * Enumera status de validacao de certificados digitais pelo
 *   Servico de Identificacao do Cidadao (SICid).
 * @author Robson Martins (robson@robsonmartins.com)
 */
@XmlType(name="status")
@XmlEnum
public enum CertificadoStatus {
    /** Desconhecido (ou corrompido). */
    UNKNOWN,
    /** Valido. */
    VALID,
    /** Expirado. */
    EXPIRED,
    /** Revogado. */
    REVOKED,
    /** Invalido (auto-assinado ou assinado por AC desconhecida). */
    INVALID;
}
```

sicid.bean.Cidadao.java

```
package sicid.bean;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Representa um cidadao, persistido via JPA e usado
 *   por metodos do servico SICid como representacao XML.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@Entity
@XmlRootElement(name="cidadao")
@SuppressWarnings("serial")
public class Cidadao implements Serializable, Cloneable {
    @Id
    private String dname;
    @Column(nullable=false)
    private String nome;
    @Column(nullable=true)
    private String nomePai;
    @Column(nullable=true)
    private String nomeMae;
    @Column(nullable=false)
    @Temporal(TemporalType.DATE)
    private Date dataNascimento;
    @Column(length=11)
    private String ric;
    @Column(length=11,nullable=true)
    private String cpf;
    @Embedded
    @Column(nullable=true)
    private DocumentoRG rg;
    @Embedded
    @Column(nullable=true)
    private DocumentoTitulo titulo;
    @Column(length=11,nullable=true)
    private String pisPasep;
```

sicid.bean.Cidadao.java

```
@Column(length=12,nullable=true)
private String cei;
@Transient
private String email;
@Transient
private String login;

/* Cria um clone do objeto corrente.
 * @return Clone do objeto corrente, ou null se houve erro. */
@Override
public Cidadao clone() {
    try {
        return (Cidadao) super.clone();
    } catch (CloneNotSupportedException e) {
        return null;
    }
}

/* Retorna uma representacao String do objeto corrente.
 * @return Representacao do objeto como String. */
@Override
public String toString() {
    StringBuilder strBuilder = new StringBuilder();
    strBuilder.append(Cidadao.class.getSimpleName())
        .append("[")
        .append(String.format("nome='%s',",nome))
        .append(String.format("nomePai='%s',",nomePai))
        .append(String.format("nomeMae='%s',",nomeMae))
        .append(String.format("dataNascimento='%s',",
                new SimpleDateFormat("dd/MM/yyyy").
                format(dataNascimento)))
        .append(String.format("ric='%s',",ric))
        .append(String.format("cpf='%s',",cpf))
        .append(String.format("rg='%s',",rg))
        .append(String.format("titulo='%s',",titulo))
        .append(String.format("pisPasep='%s',",pisPasep))
        .append(String.format("cei='%s',",cei))
        .append(String.format("email='%s',",email))
        .append(String.format("login='%s'",login))
        .append("]");
    return strBuilder.toString();
}
```

sicid.bean.Cidadao.java

```
/**
 * Retorna o Distinguished Name (DN) do certificado
 *   digital do cidadao.
 * @return DN do certificado.
 */
public String getDname() {
    return dname;
}

/**
 * Configura o Distinguished Name (DN) do certificado
 *   digital do cidadao.
 * @param dname DN do certificado.
 */
public void setDname(String dname) {
    this.dname = dname;
}

/**
 * Retorna o nome completo do cidadao.
 * @return Nome do cidadao.
 */
public String getNome() {
    return nome;
}

/**
 * Configura o nome completo do cidadao.
 * @param nome Nome do cidadao.
 */
public void setNome(String nome) {
    this.nome = nome;
}

/**
 * Retorna o nome completo do pai.
 * @return Nome do pai.
 */
public String getNomePai() {
    return nomePai;
}
```

sicid.bean.Cidadao.java

```
    /**
     * Configura o nome completo do pai.
     * @param nomePai Nome do pai.
     */
    public void setNomePai(String nomePai) {
        this.nomePai = nomePai;
    }

    /**
     * Retorna o nome completo da mae.
     * @return Nome da mae.
     */
    public String getNomeMae() {
        return nomeMae;
    }

    /**
     * Configura o nome completo da mae.
     * @param nomeMae Nome da mae.
     */
    public void setNomeMae(String nomeMae) {
        this.nomeMae = nomeMae;
    }

    /**
     * Retorna a data de nascimento.
     * @return Data de nascimento.
     */
    public Date getDataNascimento() {
        return dataNascimento;
    }

    /**
     * Configura a data de nascimento.
     * @param dataNascimento Data de nascimento.
     */
    public void setDataNascimento(Date dataNascimento) {
        this.dataNascimento = dataNascimento;
    }

    /**
     * Retorna o numero do Registro de Identidade Civil (RIC).
     * @return Numero do RIC.
     */
    public String getRic() {
        return ric;
    }
```

sicid.bean.Cidadao.java

```
/**
 * Configura o numero do Registro de Identidade Civil (RIC).
 * @param ric Numero do RIC.
 */
public void setRic(String ric) {
    this.ric = ric;
}

/**
 * Retorna o numero do Cadastro de Pessoa Fisica (CPF).
 * @return Numero do CPF.
 */
public String getCpf() {
    return cpf;
}

/**
 * Configura o numero do Cadastro de Pessoa Fisica (CPF).
 * @param cpf Numero do CPF.
 */
public void setCpf(String cpf) {
    this.cpf = cpf;
}

/**
 * Retorna os dados do Registro Geral (RG).
 * @return Objeto com os dados do RG.
 */
public DocumentoRG getRg() {
    return rg;
}
/**
 * Configura os dados do Registro Geral (RG).
 * @param rg Objeto com os dados do RG.
 */
public void setRg(DocumentoRG rg) {
    this.rg = rg;
}

/**
 * Retorna os dados do Titulo Eleitoral.
 * @return Objeto com os dados do Titulo Eleitoral.
 */
public DocumentoTitulo getTitulo() {
    return titulo;
}
```

sicid.bean.Cidadao.java

```
    /**
     * Configura os dados do Titulo Eleitoral.
     * @param titulo Objeto com os dados do Titulo Eleitoral.
     */
    public void setTitulo(DocumentoTitulo titulo) {
        this.titulo = titulo;
    }

    /**
     * Retorna o numero de inscricao no Programa de Integracao Social (PIS)
     *   ou no Programa de Formacao do Patrimonio do Servidor Publico (PASEP).
     * @return Numero do PIS/PASEP.
     */
    public String getPisPasep() {
        return pisPasep;
    }

    /**
     * Configura o numero de inscricao no Programa de Integracao Social (PIS)
     *   ou no Programa de Formacao do Patrimonio do Servidor Publico (PASEP).
     * @param pisPasep Numero do PIS/PASEP.
     */
    public void setPisPasep(String pisPasep) {
        this.pisPasep = pisPasep;
    }

    /**
     * Retorna o numero do Cadastro Especifico do INSS (CEI).
     * @return Numero do CEI.
     */
    public String getCei() {
        return cei;
    }

    /**
     * Configura o numero do Cadastro Especifico do INSS (CEI).
     * @param cei Numero do CEI.
     */
    public void setCei(String cei) {
        this.cei = cei;
    }
```

sicid.bean.Cidadao.java

```
    /**
     * Retorna o endereco de e-mail do cidadao.<br>
     *   (Atributo nao persistido em banco de dados).
     * @return Endereco de e-mail.
     */
    public String getEmail() {
        return email;
    }

    /**
     * Configura o endereco de e-mail do cidadao.
     * @param email Endereco de e-mail.
     */
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * Retorna o nome de login do cidadao.<br>
     *   (Atributo nao persistido em banco de dados).
     * @return Nome de login (username).
     */
    public String getLogin() {
        return login;
    }

    /**
     * Configura o nome de login do cidadao.
     * @param login Nome de login (username).
     */
    public void setLogin(String login) {
        this.login = login;
    }
}
```

sicid.bean.DocumentoRG.java

```
package sicid.bean;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Representa os dados de um documento RG, persistidos via JPA e usados
 *   por metodos do servico SICid como representacao XML.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@Embeddable
@XmlRootElement(name="rg")
@SuppressWarnings("serial")
public class DocumentoRG implements Serializable, Cloneable {
    @Column(name="rg",length=15,nullable=false)
    private String numero;
    @Column(name="rgOrgExpedidor",length=4,nullable=false)
    private String orgExpedidor;
    @Column(name="rgUF",length=2,nullable=false)
    private String uf;

    /* Cria um clone do objeto corrente.
     * @return Clone do objeto corrente, ou null se houve erro. */
    @Override
    public DocumentoRG clone() {
        try {
            return (DocumentoRG) super.clone();
        } catch (CloneNotSupportedException e) {
            return this;
        }
    }

    /* Retorna uma representacao String do objeto corrente.
     * @return Representacao do objeto como String. */
    @Override
    public String toString() {
        StringBuilder strBuilder = new StringBuilder();
        strBuilder.append("RG[")
            .append(String.format("numero='%s',",numero))
            .append(String.format("orgExpedidor='%s',",orgExpedidor))
            .append(String.format("uf='%s'",uf))
            .append("]");
        return strBuilder.toString();
    }
```

sicid.bean.DocumentoRG.java

```
/**
 * Retorna o numero do RG.
 * @return Numero do RG.
 */
public String getNumero() {
    return numero;
}

/**
 * Configura o numero do RG.
 * @param numero Numero do RG.
 */
public void setNumero(String numero) {
    this.numero = numero;
}

/**
 * Retorna o Orgao Expedidor do RG.
 * @return Orgao Expedidor do RG.
 */
public String getOrgExpedidor() {
    return orgExpedidor;
}

/**
 * Configura o Orgao Expedidor do RG.
 * @param orgExpedidor Orgao Expedidor do RG.
 */
public void setOrgExpedidor(String orgExpedidor) {
    this.orgExpedidor = orgExpedidor;
}

/**
 * Retorna a Unidade da Federacao (UF) do RG.
 * @return UF de expedicao RG.
 */
public String getUf() {
    return uf;
}
```

sicid.bean.DocumentoRG.java

```
    /**
     * Configura a Unidade da Federacao (UF) do RG.
     * @param uf UF de expedicao RG.
     */
    public void setUf(String uf) {
        this.uf = uf;
    }
}
```

sicid.bean.DocumentoTitulo.java

```
package sicid.bean;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Representa os dados de um documento Titulo Eleitoral,
 *   persistidos via JPA e usados por metodos do servico
 *   SICid como representacao XML.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@Embeddable
@XmlRootElement(name="titulo")
@SuppressWarnings("serial")
public class DocumentoTitulo implements Serializable, Cloneable {
    @Column(name="titulo",length=12,nullable=false)
    private String numero;
    @Column(name="tituloZona",length=3,nullable=false)
    private String zona;
    @Column(name="tituloSecao",length=4,nullable=false)
    private String secao;
    @Column(name="tituloMunicipio",length=20,nullable=false)
    private String municipio;
    @Column(name="tituloUF",length=2,nullable=false)
    private String uf;

    /* Cria um clone do objeto corrente.
     * @return Clone do objeto corrente, ou null se houve erro. */
    @Override
    public DocumentoTitulo clone() {
        try {
            return (DocumentoTitulo) super.clone();
        } catch (CloneNotSupportedException e) {
            return this;
        }
    }

    /* Retorna uma representacao String do objeto corrente.
     * @return Representacao do objeto como String. */
    @Override
    public String toString() {
        StringBuilder strBuilder = new StringBuilder();
        strBuilder.append("Titulo[")
            .append(String.format("numero='%s',",numero))
            .append(String.format("zona='%s',",zona))
```

sicid.bean.DocumentoTitulo.java

```
            .append(String.format("secao='%s',",secao))
            .append(String.format("municipio='%s',",municipio))
            .append(String.format("uf='%s'",uf))
            .append("]");
        return strBuilder.toString();
    }

    /**
     * Retorna o numero do Titulo Eleitoral.
     * @return Numero do Titulo.
     */
    public String getNumero() {
        return numero;
    }

    /**
     * Configura o numero do Titulo Eleitoral.
     * @param numero Numero do Titulo.
     */
    public void setNumero(String numero) {
        this.numero = numero;
    }

    /**
     * Retorna a Zona do Titulo Eleitoral.
     * @return Zona Eleitoral do Titulo.
     */
    public String getZona() {
        return zona;
    }

    /**
     * Configura a Zona do Titulo Eleitoral.
     * @param zona Zona Eleitoral do Titulo.
     */
    public void setZona(String zona) {
        this.zona = zona;
    }

    /**
     * Retorna a Secao do Titulo Eleitoral.
     * @return Secao Eleitoral do Titulo.
     */
    public String getSecao() {
        return secao;
    }
```

sicid.bean.DocumentoTitulo.java

```
    /**
     * Configura a Secao do Titulo Eleitoral.
     * @param secao Secao Eleitoral do Titulo.
     */
    public void setSecao(String secao) {
        this.secao = secao;
    }

    /**
     * Retorna o Municipio do Titulo Eleitoral.
     * @return Municipio do Titulo.
     */
    public String getMunicipio() {
        return municipio;
    }

    /**
     * Configura o Municipio do Titulo Eleitoral.
     * @param municipio Municipio do Titulo.
     */
    public void setMunicipio(String municipio) {
        this.municipio = municipio;
    }

    /**
     * Retorna a Unidade da Federacao (UF) do Titulo Eleitoral.
     * @return UF do Titulo Eleitoral.
     */
    public String getUf() {
        return uf;
    }

    /**
     * Configura a Unidade da Federacao (UF) do Titulo Eleitoral.
     * @param uf UF do Titulo Eleitoral.
     */
    public void setUf(String uf) {
        this.uf = uf;
    }
}
```

sicid.bean.Usuario.java

```
package sicid.bean;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

/**
 * Representa um usuario do sistema, persistido via JPA.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@Entity
@SuppressWarnings("serial")
public class Usuario implements Serializable {
    @Id
    private String dname;
    @Column(nullable=false)
    private String username;
    private String name;
    private String role;

    /**
     * Retorna o Distinguished Name (DN).
     * @return DN do certificado do usuario.
     */
    public String getDname() {
        return dname;
    }

    /**
     * Configura o Distinguished Name (DN).
     * @param dname DN do certificado do usuario.
     */
    public void setDname(String dname) {
        this.dname = dname;
    }

    /**
     * Retorna o username.
     * @return Username do usuario.
     */
    public String getUsername() {
        return username;
    }
```

sicid.bean.Usuario.java

```
    /**
     * Configura o username.
     * @param username Username do usuario.
     */
    public void setUsername(String username) {
        this.username = username;
    }

    /**
     * Retorna o nome.
     * @return Nome do usuario.
     */
    public String getName() {
        return name;
    }

    /**
     * Configura o nome.
     * @param name Nome do usuario.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Retorna a role.
     * @return Role associada ao usuario.
     */
    public String getRole() {
        return role;
    }

    /**
     * Configura a role.
     * @param role Role associada ao usuario.
     */
    public void setRole(String role) {
        this.role = role;
    }
}
```

com.robsonmartins.fiap.tcc.dao.GenericDAO.java

```
package com.robsonmartins.fiap.tcc.dao;

import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.List;
import javax.persistence.EntityManager;

/**
 * Classe abstrata com operacoes basicas para mainupulacao
 * de dados (objetos persistidos) via JPA
 * @param <T> Classe do bean persistido
 * @author Robson Martins (robson@robsonmartins.com)
 */
public abstract class GenericDAO <T> {

    /* entity manager do JPA */
    protected EntityManager em;
    /* armazena a classe concreta derivada deste DAO */
    private Class<T> classe;

    /**
     * Cria uma nova instancia de DAO.
     * @param entityManager Objeto {@link EntityManager} da API JPA.
     */
    @SuppressWarnings("unchecked")
    public GenericDAO(EntityManager entityManager){
        Class<?> thisClass = getClass();
        ParameterizedType t =
            (ParameterizedType) thisClass.getGenericSuperclass();
        Type t2 = t.getActualTypeArguments()[0];
        this.classe = (Class<T>) t2;
        em = entityManager;
    }

    /**
     * Localiza um objeto persistido pelo id
     * @param id Id do objeto
     * @return Objeto persistido
     * @throws Exception
     */
    public T localizar(Object id) throws Exception {
        T obj = null;
        try {
            obj = em.find(classe, id);
        } catch (Exception e) {
            throw e;
        }
```

com.robsonmartins.fiap.tcc.dao.GenericDAO.java

```
        return obj;
    }

    /**
     * Lista todos os objetos persistidos da classe
     * @return Lista de objetos persistidos
     * @throws Exception
     */
    @SuppressWarnings("unchecked")
    public List<T> listar() throws Exception {
        List<T> list = null;
        try {
            list = (List<T>) em.createQuery(
                    "from " + classe.getSimpleName()).getResultList();
        } catch (Exception e) {
            throw e;
        }
        return list;
    }

    /**
     * Insere (persiste) um objeto
     * @param obj Objeto a ser persistido
     * @throws Exception
     */
    public synchronized void inserir(T obj) throws Exception {
        em.merge(obj);
        em.flush();
    }

    /**
     * Exclui um objeto persistido
     * @param id Id do objeto a ser removido
     * @throws Exception
     */
    public synchronized void excluir(Object id) throws Exception {
        T obj = em.find(classe, id);
        em.remove(obj);
        em.flush();
    }
}
```

sicid.dao.UsuarioDAO.java

```
package sicid.dao;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

import com.robsonmartins.fiap.tcc.dao.GenericDAO;
import sicid.bean.Usuario;

/**
 * Classe de manipulacao de objetos persistidos via JPA
 * Objetos da classe bean {@link Usuario}
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class UsuarioDAO extends GenericDAO<Usuario> {
    protected static Logger logger;
    protected static boolean trace;

    /**
     * Cria uma nova instancia de DAO.
     * @param entityManager Objeto {@link EntityManager} da API JPA.
     */
    public UsuarioDAO(EntityManager entityManager) {
        super(entityManager);
        logger = LogManager.getLogger(UsuarioDAO.class);
        trace = logger.isTraceEnabled();
    }

    /**
     * Retorna um Usuario pelo DN.
     * @param dname Distinguished Name (DN)
     *   do Usuario no registro do banco de dados.
     * @return Objeto que representa o Usuario.
     */
    public Usuario localizar(String dname) {
        if (trace) {
            logger.trace(String.format("Localizar usuario DN: %s", dname));
        }
        Usuario usuario = null;
        try {
            usuario = super.localizar(dname);
        } catch (Exception e) {
            if (trace) {
                logger.trace("Erro ao localizar usuario", e);
            }
```

sicid.dao.UsuarioDAO.java

```
        }
        if (usuario != null) {
            if (trace) {
                logger.trace(String.format("Usuario encontrado - DN: %s",
                        dname));
            }
        } else {
            if (trace) {
                logger.trace("Usuario nao encontrado");
            }
        }
        return usuario;
    }

    /**
     * Retorna um Usuario pelo Username.
     * @param username Username do usuario.
     * @return Objeto que representa o Usuario.
     */
    public Usuario localizarPorUsername(String username) {
        if (username == null) { return null; }
        if (trace) {
            logger.trace(String.format(
                    "Localizar usuario username='%s'", username));
        }
        Usuario usuario = null;
        Query q = null;
        try {
            q = em.createQuery("from Usuario where username like :username");
            q.setParameter("username", username);
            usuario = (Usuario) q.getSingleResult();
        } catch (Exception e) {
            if (trace) {
                logger.trace(String.format(
                        "Localizar usuario: %s",e.getLocalizedMessage()));
            }
        }
        return usuario;
    }

    /**
     * Lista todos os Usuarios.
     * @return Lista de Usuarios.
     */
    @Override
    public List<Usuario> listar() {
        if (trace) {
```

sicid.dao.UsuarioDAO.java

```
            logger.trace("Listar usuarios");
        }
        List<Usuario> list = null;
        try {
            list = super.listar();
        } catch (Exception e) {
            if (trace) {
                logger.trace("Erro ao Listar usuario", e);
            }
        }
        if (trace) {
            logger.trace(String.format("%d usuarios encontrados",
                    (list != null) ? list.size() : 0));
        }
        return list;
    }

    /**
     * Lista Usuarios por Role.
     * @param role Role dos usuarios.
     * @return Lista de Usuarios.
     */
    @SuppressWarnings("unchecked")
    public List<Usuario> listarPorRole(String role) {
        if (trace) {
            logger.trace(String.format(
                    "Listar usuarios por role=%s", role));
        }
        List<Usuario> list = null;
        Query q = null;
        try {
            q = em.createQuery("from Usuario where role like :role");
            q.setParameter("role", role);
            list = (List<Usuario>) q.getResultList();
        } catch (Exception e) {
            if (trace) {
                logger.trace("Erro ao Listar usuarios", e);
            }
        }

        if (trace) {
            logger.trace(String.format("%d usuarios encontrados",
                    (list != null) ? list.size() : 0));
        }
        return list;
    }
```

sicid.dao.UsuarioDAO.java

```
    /**
     * Insere ou Atualiza (persiste) um Usuario.
     * @param usuario Usuario a ser persistido.
     * @throws Exception
     */
    @Override
    public void inserir(Usuario usuario) throws Exception {
        if (trace) {
            logger.trace(String.format("Inserir/Atualizar usuario: %s",
                    usuario.getName()));
        }
        try {
            super.inserir(usuario);
            if (trace) {
                logger.trace("Usuario inserido/atualizado");
            }
        } catch (Exception e) {
            logger.error("Erro ao inserir/atualizar usuario", e);
            throw new Exception("Erro ao inserir/atualizar usu\u00E1rio", e);
        }
    }

    /**
     * Exclui um Usuario do banco de dados.
     * @param dname Distinguished Name (DN)
     *   do Usuario a ser removido.
     * @throws Exception
     */
    public void excluir(String dname) throws Exception {
        if (trace) {
            logger.trace(String.format("Excluir usuario DN: %s", dname));
        }
        try {
            super.excluir(dname);
            if (trace) {
                logger.trace("Usuario excluido");
            }
        } catch (Exception e) {


            logger.error("Erro ao excluir usuario", e);
            throw new Exception("Erro ao excluir usu\u00E1rio", e);
        }
    }
}
```

sicid.model.ISICidEngine.java

```
package sicid.model;

import java.io.InputStream;
import java.util.List;
import javax.ejb.Local;

import sicid.bean.CertificadoConfiavel;
import sicid.bean.CertificadoStatus;
import sicid.bean.Cidadao;
import sicid.bean.ConsumidorConfiavel;
import sicid.bean.Usuario;

/**
 * Motor do Servico de Identificacao do Cidadao (SICid).<br/>
 * Implementa as funcionalidades do servico.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@Local
public interface ISICidEngine {
    /** Role para acessar o servico SICid. */
    public static final String SICID_ACCESS_ROLE = "wsuser";
    /** Role para acessar a aplicacao web do SICid Admin. */
    public static final String SICIDWEB_ACCESS_ROLE = "admin";

    /**
     * Valida um certificado digital.
     * @param content Conteudo de um certificado, codificado em Base64.
     * @return Status do certificado.
     */
    public CertificadoStatus validarCertificado(String content);

    /**
     * Consulta os dados cadastrais de um cidadao,
     *   a partir de seu certificado digital.
     * @param content Conteudo de um certificado, codificado em Base64.
     * @return Dados cadastrais de um cidadao.
     */
    public Cidadao consultarCidadao(String content);

    /**
     * Retorna a lista de cidadaos cadastrados.
     * @return Lista de cidadaos.
     */
    public List<Cidadao> listarCidadaos();
```

sicid.model.ISICidEngine.java

```
/**
 * Retorna a lista de certificados confiaveis cadastrados.
 * @return Lista de certificados confiaveis.
 */
public List<CertificadoConfiavel> listarCertConfiaveis();

/**
 * Adiciona um certificado confiavel ao cadastro, a partir de um objeto
 *   {@link InputStream} (que pode apontar para um arquivo de certificado).
 * @param istream Objeto InputStream.
 * @throws Exception
 */
public void adicionarCertConfiavel(InputStream istream) throws Exception;

/**
 * Remove um certificado confiavel do cadastro.
 * @param id Id do certificado no banco de dados.
 * @throws Exception
 */
public void removerCertConfiavel(long id) throws Exception;

/**
 * Retorna a lista de consumidores confiaveis cadastrados.
 * @return Lista de consumidores confiaveis.
 */
public List<ConsumidorConfiavel> listarAppsConfiaveis();

/**
 * Adiciona uma aplicacao confiavel ao cadastro, a partir de um objeto
 *   {@link InputStream} (que pode apontar para um arquivo de certificado).
 * @param istream Objeto InputStream.
 * @throws Exception
 */
public void adicionarAppConfiavel(InputStream istream) throws Exception;

/**
 * Remove um consumidor confiavel do cadastro.
 * @param dname Distinguished Name (DN) do consumidor no banco de dados.
 * @throws Exception
 */
public void removerAppConfiavel(String dname) throws Exception;
```

sicid.model.ISICidEngine.java

```
/**
 * Adiciona um usuario administrador ao cadastro, a partir de um objeto
 *   {@link InputStream} (que pode apontar para um arquivo de certificado).
 * @param istream Objeto InputStream.
 * @throws Exception
 */
public void adicionarAdministrador(InputStream istream) throws Exception;

/**
 * Remove um usuario do cadastro.
 * @param dname Distinguished Name (DN) do usuario no banco de dados.
 * @throws Exception
 */
public void removerUsuario(String dname) throws Exception;

/**
 * Retorna a lista de usuarios cadastrados.
 * @return Lista de usuarios.
 */
public List<Usuario> listarUsuarios();

/**
 * Retorna a lista de usuarios cadastrados, para a role especificada.
 * @param role Role dos usuarios.
 * @return Lista de usuarios.
 */
public List<Usuario> listarUsuariosPorRole(String role);

/**
 * Retorna um Usuario pelo DN.
 * @param dname Distinguished Name (DN)
 *   do Usuario no registro do banco de dados.
 * @return Usuario ou null se nao encontrado.
 */
public Usuario localizarUsuario(String dname);

/**
 * Adiciona uma cidadao ao cadastro.
 * @param cidadao Objeto com os atributos do cidadao.
 * @throws Exception
 */
public void adicionarCidadao(Cidadao cidadao) throws Exception;
```

sicid.model.ISICidEngine.java

```
    /**
     * Remove um cidadao do cadastro.
     * @param dname Distinguished Name (DN) do cidadao no banco de dados.
     * @throws Exception
     */
    public void removerCidadao(String dname) throws Exception;

    /**
     * Retorna informacoes de um cidadao contidas em seu certificado digital.
     * @param content Conteudo de um certificado, codificado em Base64.
     * @return Objeto com as informacoes de um cidadao.
     * @throws Exception
     */
    public Cidadao getCidadaoInfoFromCert(String content) throws Exception;
}
```

sicid.model.SICidEngine.java

```
package sicid.model;

import java.io.InputStream;
import java.security.MessageDigest;
import java.security.cert.X509Certificate;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Map;
import javax.annotation.PostConstruct;
import javax.annotation.security.PermitAll;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import sun.misc.BASE64Encoder;

import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;
import sicid.util.CertificadoIcpBrasilParser;
import sicid.util.CertificadoIcpBrasilParser.AtributoIcpBrasil;
import sicid.util.CertificadoValidador;
import sicid.bean.CertificadoConfiavel;
import sicid.bean.CertificadoStatus;
import sicid.bean.Cidadao;
import sicid.bean.ConsumidorConfiavel;
import sicid.bean.DocumentoRG;
import sicid.bean.DocumentoTitulo;
import sicid.bean.Usuario;
import sicid.dao.CertificadoConfiavelDAO;
import sicid.dao.CidadaoDAO;
import sicid.dao.ConsumidorConfiavelDAO;
import sicid.dao.UsuarioDAO;

/**
 * Motor do Servico de Identificacao do Cidadao (SICid).<br/>
 * Implementa as funcionalidades do servico.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@PermitAll
@Stateless
public class SICidEngine implements ISICidEngine {
    /* nome da unidade de persistencia configurada em persistence.xml */
    private static final String PERSISTENCE_UNIT_NAME = "sicid";
    /* DAO de certificados confiaveis */
    private CertificadoConfiavelDAO certConfiavelDAO;
    /* DAO de consumidores confiaveis */
    private ConsumidorConfiavelDAO consConfiavelDAO;
```

sicid.model.SICidEngine.java

```
/* DAO de usuarios */
private UsuarioDAO usuarioDAO;
/* DAO de cidadaos */
private CidadaoDAO cidadaoDAO;
/* EntityManager para JPA. */
@PersistenceContext(unitName=PERSISTENCE_UNIT_NAME)
private EntityManager entityManager;
/* para fazer log */
private static Logger logger;
private static boolean trace;

/**
 * Cria uma nova instancia do motor do SICid.
 */
public SICidEngine() {
    logger = LogManager.getLogger(SICidEngine.class);
    trace = logger.isTraceEnabled();
}

/* inicializa DAO's */
@PostConstruct
protected void init() {
    certConfiavelDAO = new CertificadoConfiavelDAO(entityManager);
    consConfiavelDAO = new ConsumidorConfiavelDAO(entityManager);
    usuarioDAO = new UsuarioDAO(entityManager);
    cidadaoDAO = new CidadaoDAO(entityManager);
}

@Override
public CertificadoStatus validarCertificado(String content) {
    X509Certificate x509Cert = null;
    try {
        if (trace) {
            logger.trace("Validando um certificado");
        }
        if (content == null) {
            throw new NullPointerException(
                "Certificado inv\u00E1lido (nulo).");
        }
        x509Cert = CertificadoSerializador.strToCert(content);
        if (trace) {
            logger.trace(String.format("Certificado (DN): %s",
                    x509Cert.getSubjectX500Principal().getName()));
        }
```

sicid.model.SICidEngine.java

```
            if (CertificadoValidador.isSelfSigned(x509Cert)) {
                if (trace) {
                    logger.trace("Certificado INVALIDO: auto-assinado");
                }
                return CertificadoStatus.INVALID;
            }
            if (!CertificadoValidador.isValidByDate(x509Cert)) {
                if (trace) {
                    logger.trace("Certificado EXPIRADO");
                }
                return CertificadoStatus.EXPIRED;
            }
            if (!CertificadoValidador.isValidKeyChain(x509Cert,
                    listarX509CertConfiaveis())) {
                if (trace) {
                    logger.trace("Certificado INVALIDO: cadeia nao-confiavel");
                }
                return CertificadoStatus.INVALID;
            }
            if (CertificadoValidador.isRevoked(x509Cert)) {
                if (trace) {
                    logger.trace("Certificado REVOGADO");
                }
                return CertificadoStatus.REVOKED;
            }
            if (trace) {
                logger.trace("Certificado VALIDO");
            }
            return CertificadoStatus.VALID;

        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao validar certificado", e);
            }
            return CertificadoStatus.UNKNOWN;
        }
    }

    @Override
    public Cidadao consultarCidadao(String content) {
        X509Certificate x509Cert = null;
        Cidadao cidadao = null;
        Map<AtributoIcpBrasil, String> props = null;
        try {
            if (trace) {
                logger.trace("Consultando informacoes de um cidadao");
            }
```

sicid.model.SICidEngine.java

```
            if (content == null) {
                throw new NullPointerException(
                    "Certificado inv\u00E1lido (nulo).");
            }
            x509Cert = CertificadoSerializador.strToCert(content);
            if (trace) {
                logger.trace(String.format("Certificado (DN): %s",
                        x509Cert.getSubjectX500Principal().getName()));
            }
            cidadao =
                cidadaoDAO.localizar(
                    x509Cert.getSubjectX500Principal().getName());
            if (cidadao == null) {
                throw new Exception("Cidad\u00E3o n\u00E3o encontrado.");
            }
            props =
                CertificadoIcpBrasilParser.getAtributosIcpBrasil(x509Cert);
            if (props != null) {
                cidadao.setEmail(props.get(AtributoIcpBrasil.EMAIL));
                cidadao.setLogin(props.get(AtributoIcpBrasil.LOGIN));
            }
            if (trace) {
                logger.trace(String.format(
                        "Obtidas informacoes do cidadao '%s'",
                        cidadao.getNome()));
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao obter informacoes do cidadao", e);
            }
            cidadao = new Cidadao();
        }
        return cidadao;
    }

    @Override
    public List<Cidadao> listarCidadaos() {
        return cidadaoDAO.listar();
    }

    @Override
    public List<CertificadoConfiavel> listarCertConfiaveis() {
        return certConfiavelDAO.listar();
    }
```

sicid.model.SICidEngine.java

```
@Override
public void adicionarCertConfiavel(InputStream istream) throws Exception {
    try {
        if (trace) {
            logger.trace("Adicionando certificado confiavel");
        }
        X509Certificate x509cert =
            CertificadoSerializador.loadCertFromStream(istream);
        CertificadoConfiavel trustedCert = new CertificadoConfiavel();
        trustedCert.setId(0);
        trustedCert.setX509Certificate(x509cert);
        certConfiavelDAO.inserir(trustedCert);
        if (trace) {
            logger.trace("Certificado confiavel adicionado com sucesso");
        }
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao adicionar certificado confiavel", e);
        }
        throw e;
    }
}

@Override
public void removerCertConfiavel(long id) throws Exception {
    try {
        if (trace) {
            logger.trace("Removendo certificado confiavel");
        }
        certConfiavelDAO.excluir(id);
        if (trace) {
            logger.trace("Certificado confiavel removido com sucesso");
        }
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao remover certificado confiavel", e);
        }
        throw e;
    }
}

@Override
public List<ConsumidorConfiavel> listarAppsConfiaveis() {
    return consConfiavelDAO.listar();
}
```

sicid.model.SICidEngine.java

```
@Override
public void adicionarAppConfiavel(InputStream istream) throws Exception {
    try {
        if (trace) {
            logger.trace("Adicionando aplicacao confiavel");
        }
        X509Certificate x509cert =
            CertificadoSerializador.loadCertFromStream(istream);
        ConsumidorConfiavel trustedApp = new ConsumidorConfiavel();
        String dname = x509cert.getSubjectX500Principal().getName();
        trustedApp.setDname(dname);
        MessageDigest md = MessageDigest.getInstance("SHA1");
        String id =
            new BASE64Encoder().encode(
                    md.digest(dname.getBytes()));
        trustedApp.setId(id);
        String name = CertificadoIcpBrasilParser.extractNomeFromCN(dname);
        if (name == null) { name = dname; }
        trustedApp.setName(name);
        trustedApp.setRole(SICID_ACCESS_ROLE);
        consConfiavelDAO.inserir(trustedApp);
        if (trace) {
            logger.trace("Aplicacao confiavel adicionada com sucesso");
        }
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao adicionar aplicacoa confiavel", e);
        }
        throw e;
    }
}

@Override
public void removerAppConfiavel(String dname) throws Exception {
    try {
        if (trace) {
            logger.trace("Removendo aplicacao confiavel");
        }
        consConfiavelDAO.excluir(dname);
        if (trace) {
            logger.trace("Aplicacao confiavel removido com sucesso");
        }
```

sicid.model.SICidEngine.java

```
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao remover aplicacao confiavel", e);
            }
            throw e;
        }
    }

    @Override
    public void adicionarAdministrador(InputStream istream) throws Exception {
        try {
            if (trace) {
                logger.trace("Adicionando administrador");
            }
            X509Certificate x509cert =
                CertificadoSerializador.loadCertFromStream(istream);
            Usuario trustedUser = new Usuario();
            String dname = x509cert.getSubjectX500Principal().getName();
            trustedUser.setDname(dname);
            MessageDigest md = MessageDigest.getInstance("SHA1");
            String id =
                new BASE64Encoder().encode(
                        md.digest(dname.getBytes()));
            trustedUser.setUsername(id);
            String name = CertificadoIcpBrasilParser.extractNomeFromCN(dname);
            if (name == null) { name = dname; }
            trustedUser.setName(name);
            trustedUser.setRole(SICIDWEB_ACCESS_ROLE);
            usuarioDAO.inserir(trustedUser);
            if (trace) {
                logger.trace("Administrador adicionado com sucesso");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao adicionar administrador", e);
            }
            throw e;
        }
    }

    @Override
    public void removerUsuario(String dname) throws Exception {
        try {
            if (trace) {
                logger.trace("Removendo usuario");
            }
            usuarioDAO.excluir(dname);
```

sicid.model.SICidEngine.java

```
            if (trace) {
                logger.trace("Usuario removido com sucesso");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao remover usuario", e);
            }
            throw e;
        }
    }

    @Override
    public List<Usuario> listarUsuarios() {
        return usuarioDAO.listar();
    }

    @Override
    public List<Usuario> listarUsuariosPorRole(String role) {
        return usuarioDAO.listarPorRole(role);
    }

    @Override
    public Usuario localizarUsuario(String dname) {
        return usuarioDAO.localizar(dname);
    }

    @Override
    public void adicionarCidadao(Cidadao cidadao) throws Exception {
        try {
            if (trace) {
                logger.trace("Adicionando cidadao");
            }
            cidadaoDAO.inserir(cidadao);
            if (trace) {
                logger.trace("Cidadao adicionado com sucesso");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao adicionar cidadao", e);
            }
            throw e;
        }
    }
```

sicid.model.SICidEngine.java

```
@Override
public void removerCidadao(String dname) throws Exception {
    try {
        if (trace) {
            logger.trace("Removendo cidadao");
        }
        cidadaoDAO.excluir(dname);
        if (trace) {
            logger.trace("Cidadao removido com sucesso");
        }
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao remover cidadao", e);
        }
        throw e;
    }
}

@Override
public Cidadao getCidadaoInfoFromCert(String content) throws Exception {
    X509Certificate cert = null;
    Map<AtributoIcpBrasil, String> props = null;
    Cidadao cidadao = new Cidadao();
    DocumentoRG rg = new DocumentoRG();
    DocumentoTitulo titulo = new DocumentoTitulo();
    if (trace) {
        logger.trace("Obtendo informacoes de um certificado");
    }
    try {
        cert = CertificadoSerializador.strToCert(content);
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao obter certificado: conteudo invalido", e);
        }
        throw e;
    }
    if (trace) {
        logger.trace(String.format("Certificado (DN): %s",
                cert.getSubjectX500Principal().getName()));
    }
```

sicid.model.SICidEngine.java

```
        try {
            props = CertificadoIcpBrasilParser.getAtributosIcpBrasil(cert);
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao obter informacoes do certificado", e);
            }
            throw e;
        }
        if (props != null) {
            cidadao.setDname(cert.getSubjectX500Principal().getName());
            cidadao.setNome(props.get(AtributoIcpBrasil.NOME_RESPONSAVEL));
            try {
                cidadao.setDataNascimento(
                        new SimpleDateFormat("ddMMyyyy").parse(
                                props.get(AtributoIcpBrasil.NASCIMENTO)));
            } catch (Exception e) {
                if (trace) {
                    logger.trace("Erro ao obter data de nascimento", e);
                }
            }
            cidadao.setRic(props.get(AtributoIcpBrasil.RIC));
            cidadao.setCpf(props.get(AtributoIcpBrasil.CPF));
            rg.setNumero(props.get(AtributoIcpBrasil.RG));
            rg.setOrgExpedidor(props.get(AtributoIcpBrasil.RG_ORGEXPEDIDOR));
            rg.setUf(props.get(AtributoIcpBrasil.RG_UF));
            titulo.setNumero(props.get(AtributoIcpBrasil.TITULO));
            titulo.setZona(props.get(AtributoIcpBrasil.TITULO_ZONA));
            titulo.setSecao(props.get(AtributoIcpBrasil.TITULO_SECAO));
            titulo.setMunicipio(props.get(AtributoIcpBrasil.TITULO_MUNICIPIO));
            titulo.setUf(props.get(AtributoIcpBrasil.TITULO_UF));
            cidadao.setPisPasep(props.get(AtributoIcpBrasil.PISPASEP));
            cidadao.setCei(props.get(AtributoIcpBrasil.CEI));
            cidadao.setEmail(props.get(AtributoIcpBrasil.EMAIL));
            cidadao.setLogin(props.get(AtributoIcpBrasil.LOGIN));
        }
        cidadao.setRg(rg);
        cidadao.setTitulo(titulo);
        if (trace) {
            logger.trace(String.format(
                    "Obtidas informacoes do cidadao '%s'", cidadao.getNome()));
        }
        return cidadao;
    }
```

sicid.model.SICidEngine.java

```
    /* Retorna uma lista de objetos {@link X509Certificate}, que
     *   representam os certificados confiaveis cadastrados.
     * @return Lista de certificados confiaveis, como objetos
     *   {@link X509Certificate}.
     */
    private X509Certificate[] listarX509CertConfiaveis() {
        List<CertificadoConfiavel> trustedCerts = listarCertConfiaveis();
        X509Certificate[] trustedX509Certs =
            new X509Certificate[(trustedCerts != null) ? trustedCerts.size() : 0];
        if (trustedCerts != null) {
            int idx = 0;
            for (CertificadoConfiavel cert : trustedCerts) {
                trustedX509Certs[idx++] = cert.getX509Certificate();
            }
        }
        return trustedX509Certs;
    }
}
```

sicid.util.CertificadoValidador.java

```
package sicid.util;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.net.URL;
import java.security.KeyStore;
import java.security.PublicKey;
import java.security.cert.CRLException;
import java.security.cert.CertPath;
import java.security.cert.CertPathValidator;
import java.security.cert.CertPathValidatorException;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.PKIXParameters;
import java.security.cert.TrustAnchor;
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;
import java.util.Set;
import javax.naming.Context;
import javax.naming.directory.Attribute;
import javax.naming.directory.Attributes;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import org.bouncycastle.asn1.ASN1InputStream;
import org.bouncycastle.asn1.ASN1Primitive;
import org.bouncycastle.asn1.DERIA5String;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.X509Extension;
```

sicid.util.CertificadoValidador.java

```
/**
 * Implementa metodos para validacao de certificados digitais.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class CertificadoValidador {
    /**
     * Verifica a validade (datas de validade/expiracao) de um certificado.
     * @param cert Objeto que representa o certificado a ser validado.
     * @return True se certificado esta' valido (dentro das datas de uso).
     */
    public static boolean isValidByDate(X509Certificate cert) {
        try {
            cert.checkValidity();
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Verifica se um certificado e' auto-assinado.
     * @param cert Objeto que representa o certificado a ser validado.
     * @return True se certificado e' auto-assinado.
     */
    public static boolean isSelfSigned(X509Certificate cert) {
          try {
           PublicKey key = cert.getPublicKey();
           cert.verify(key);
           return true;
       } catch (Exception e) {
           return false;
       }
    }

    /**
     * Verifica se um certificado tem sua cadeia valida dentro de um
     *   conjunto de certificados emitentes confiaveis.
     * @param cert Objeto que representa o certificado a ser validado.
     * @param keyStore KeyStore contendo o conjunto de certificados
     *   emissores confiaveis.
     * @return True se o certificado foi emitido por um dos emissores
     *   confiaveis especificados.
     * @throws Exception
     */
    public static boolean isValidKeyChain(X509Certificate cert,
           KeyStore keyStore) throws Exception {
```

sicid.util.CertificadoValidador.java

```
        X509Certificate[] trustedCerts = new X509Certificate[keyStore.size()];
        int argIdx = 0;
        Enumeration<String> alias = keyStore.aliases();
        while (alias.hasMoreElements()) {
            trustedCerts[argIdx++] =
                (X509Certificate) keyStore.getCertificate(alias.nextElement());
        }
        return isValidKeyChain(cert, trustedCerts);
    }

    /**
     * Verifica se um certificado tem sua cadeia valida dentro de um
     *   conjunto de certificados emitentes confiaveis.
     * @param cert Objeto que representa o certificado a ser validado.
     * @param trustedCerts Conjunto de certificados emissores confiaveis.
     * @return True se o certificado foi emitido por um dos emissores
     *   confiaveis especificados.
     * @throws Exception
     */
    public static boolean isValidKeyChain(X509Certificate cert,
           X509Certificate[] trustedCerts) throws Exception {

        if (trustedCerts == null || trustedCerts.length == 0) {
            return true;
        }
        boolean found = false;
        int argIdx = trustedCerts.length;
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        CertPathValidator validator = CertPathValidator.getInstance("PKIX");
        TrustAnchor anchor;
        Set<TrustAnchor> anchors;
        CertPath certPath;
        List<Certificate> certList;
        PKIXParameters params;
        while (!found && argIdx > 0) {
            anchor = new TrustAnchor(trustedCerts[--argIdx], null);
            anchors = Collections.singleton(anchor);
            certList = Arrays.asList(new Certificate[] { cert });
            certPath = certFactory.generateCertPath(certList);
            params = new PKIXParameters(anchors);
            params.setRevocationEnabled(false);
            if (cert.getIssuerDN().equals(trustedCerts[argIdx].getSubjectDN())) {
                try {
                    validator.validate(certPath, params);
                    if (isSelfSigned(trustedCerts[argIdx])) {
                        found = true;
                    } else if (!cert.equals(trustedCerts[argIdx])) {
```

sicid.util.CertificadoValidador.java

```
                    found = isValidKeyChain(trustedCerts[argIdx],
                                            trustedCerts);
                }
            } catch (CertPathValidatorException e) { }
        }
    }
    return found;
}

/**
 * Vertifica se um certificado foi revogado pelo seu emissor
 *   (ou um emissor na sua cadeia de certificados), atraves da
 *   consulta 'as CRLs (Listas de Certificados Revogados).
 * @param cert Objeto que representa o certificado a ser validado.
 * @return True se certificado foi revogado.
 * @throws Exception
 */
public static boolean isRevoked(X509Certificate cert) throws Exception {
    for (String crlURL : getCrlDistPoints(cert)) {
        if (downloadCRL(crlURL).isRevoked(cert)) {
            return true;
        }
    }
    return false;
}

/* Retorna uma lista com os pontos de distribuicao de CRL
 *   (Lista de Certificados Revogados).
 * @param cert Objeto que representa o certificado.
 * @return Lista de pontos de distribuicao de CRL.
 * @throws Exception
 */
private static List<String> getCrlDistPoints(X509Certificate cert)
                                    throws Exception {
    byte[] crlDistPointExt =
        cert.getExtensionValue(X509Extension.cRLDistributionPoints.getId());
    if (crlDistPointExt == null) {
        List<String> emptyList = new ArrayList<String>();
        return emptyList;
    }
    ASN1InputStream asnInStream =
        new ASN1InputStream(new ByteArrayInputStream(crlDistPointExt));
    ASN1Primitive derObjCrlDistPoint = asnInStream.readObject();
    DEROctetString derOctStrCrlDistPoint =
        (DEROctetString) derObjCrlDistPoint;
    byte[] crlDistPointBytes = derOctStrCrlDistPoint.getOctets();
```

sicid.util.CertificadoValidador.java

```
        ASN1InputStream asnInStream2 =
            new ASN1InputStream(new ByteArrayInputStream(crlDistPointBytes));
        ASN1Primitive derObjCrlDistPoint2 = asnInStream2.readObject();
        CRLDistPoint crlDistPoints =
            CRLDistPoint.getInstance(derObjCrlDistPoint2);
        List<String> crlUrls = new ArrayList<String>();
        for (DistributionPoint dPoint: crlDistPoints.getDistributionPoints()) {
            DistributionPointName dPointName = dPoint.getDistributionPoint();
            if (dPointName != null) {
                if (dPointName.getType() == DistributionPointName.FULL_NAME) {
                    GeneralName[] genNames =
                        GeneralNames.getInstance(dPointName.getName()).getNames();
                    for (int j = 0; j < genNames.length; j++) {
                        if (genNames[j].getTagNo() ==
                                GeneralName.uniformResourceIdentifier) {
                            String url = DERIA5String.getInstance(
                                genNames[j].getName()).getString();
                            crlUrls.add(url);
                        }
                    }
                }
            }
        }
        return crlUrls;
    }

    /* Realiza o download de uma CRL (Lista de Certificados Revogados).
     * @param crlURL URL da CRL (Lista de Certificados Revogados).
     * @return Objeto que representa a CRL.
     * @throws Exception
     */
    private static X509CRL downloadCRL(String crlURL) throws Exception {
        if (crlURL.startsWith("http://" ) ||
            crlURL.startsWith("https://") ||
            crlURL.startsWith("ftp://"  )) {
            return downloadCRLFromWeb(crlURL);
        } else if (crlURL.startsWith("ldap://")) {
            return downloadCRLFromLDAP(crlURL);
        } else {
            throw new CRLException(
                String.format(
                   "Can not download CRL from certificate distribution point: %s",
                   crlURL));
        }
    }
```

sicid.util.CertificadoValidador.java

```
    /* Realiza o download de uma CRL (Lista de Certificados Revogados),
     *   a partir da web (HTTP, HTTPS, FTP).
     * @param crlURL URL da CRL (Lista de Certificados Revogados).
     * @return Objeto que representa a CRL.
     * @throws Exception
     */
    private static X509CRL downloadCRLFromWeb(String crlURL) throws Exception {
        URL url = new URL(crlURL);
        InputStream crlStream = url.openStream();
        try {
            CertificateFactory certFactory =
                CertificateFactory.getInstance("X.509");
            return (X509CRL) certFactory.generateCRL(crlStream);
        } finally {
            crlStream.close();
        }
    }

    /* Realiza o download de uma CRL (Lista de Certificados Revogados),
     *   a partir de um servidor LDAP.
     * @param ldapURL URL da CRL (Lista de Certificados Revogados),
     *   no servicor LDAP.
     * @return Objeto que representa a CRL.
     * @throws Exception
     */
    private static X509CRL downloadCRLFromLDAP(String ldapURL) throws Exception {
        Hashtable<String, String> env = new Hashtable<String, String>();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, ldapURL);
        DirContext context = new InitialDirContext(env);
        Attributes attrs = context.getAttributes("");
        Attribute crlAttr = attrs.get("certificateRevocationList;binary");
        byte[] crlValue = (byte[]) crlAttr.get();
        if ((crlValue == null) || (crlValue.length == 0)) {
            throw new CRLException(
                    String.format("Can not download CRL from: %s", ldapURL));
        } else {
            InputStream crlStream = new ByteArrayInputStream(crlValue);
            CertificateFactory certFactory =
                CertificateFactory.getInstance("X.509");
            X509CRL crl = (X509CRL) certFactory.generateCRL(crlStream);
            return crl;
        }
    }
}
```

sicid.util.CertificadoIcpBrasilParser.java

```
package sicid.util;

import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.ASN1Primitive;
import org.bouncycastle.asn1.ASN1Sequence;
import org.bouncycastle.asn1.ASN1TaggedObject;
import org.bouncycastle.asn1.DERObjectIdentifier;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.DERPrintableString;
import org.bouncycastle.asn1.DERUTF8String;
import org.bouncycastle.x509.extension.X509ExtensionUtil;

/**
 * Implementa metodos para extrair informacoes de certificados no padrao
 *  ICP Brasil.
 * @author Robson Martins (robson@robsonmartins.com)
 * @see <a target="_blank"
 * href="http://www.iti.gov.br/twiki/bin/view/Certificacao/EstruturaIcp">
 * Estrutura da ICP-Brasil</a>
 */
public class CertificadoIcpBrasilParser {
    /**
     * Enumera os principais atributos encontrados nos certificados
     *   emitidos no padrao ICP Brasil.
     */
    public enum AtributoIcpBrasil {
        /** Nome do Responsavel ou Titular. */
        NOME_RESPONSAVEL,
        /** Nome da Pessoa Juridica (se certificado PJ). */
        NOME_PJ,
        /** Numero do Registro de Identidade Civil (RIC). */
        RIC,
        /** Numero do Cadastro de Pessoa Fisica (CPF). */
        CPF,
        /** Numero do Cadastro Nacional de Pessoa Juridica (CNPJ). */
        CNPJ,
        /** Endereco de email do titular ou responsavel. */
        EMAIL,
        /** Data de nascimento do titular ou responsavel. */
        NASCIMENTO,
```

sicid.util.CertificadoIcpBrasilParser.java

```
        /** Numero de cadastro do Programa de Integracao Social (PIS) ou
         *   Programa de Formacao do Patrimonio do Servidor Publico (PASEP) do
         *   titular ou responsavel. */
        PISPASEP,
        /** Numero do Registro Geral (RG) do titular ou responsavel. */
        RG,
        /** Orgao Expedidor do RG do titular ou responsavel. */
        RG_ORGEXPEDIDOR,
        /** Unidade da Federacao (UF) do RG do titular ou responsavel. */
        RG_UF,
        /** Numero de inscricao do Cadastro Especifico do INSS (CEI) do
         *   titular ou responsavel. */
        CEI,
        /** Numero do Titulo Eleitoral do titular ou responsavel. */
        TITULO,
        /** Zona do Titulo Eleitoral do titular ou responsavel. */
        TITULO_ZONA,
        /** Secao do Titulo Eleitoral do titular ou responsavel. */
        TITULO_SECAO,
        /** Municipio do Titulo Eleitoral do titular ou responsavel. */
        TITULO_MUNICIPIO,
        /** Unidade da Federacao (UF) do Titulo Eleitoral
         *   do titular ou responsavel. */
        TITULO_UF,
        /** Nome de login do titular. */
        LOGIN,
        /** Globally Unique IDentifier (GUID) do servidor
         *   (se certificado destinado para equipamento). */
        GUID_SERVER;
    }
    /* OID's definidos pela ICP Brasil */
    private static final DERObjectIdentifier OID_PF_DADOS_TITULAR     =
            new DERObjectIdentifier("2.16.76.1.3.1");
    private static final DERObjectIdentifier OID_PJ_RESPONSAVEL       =
            new DERObjectIdentifier("2.16.76.1.3.2");
    private static final DERObjectIdentifier OID_PJ_CNPJ              =
            new DERObjectIdentifier("2.16.76.1.3.3");
    private static final DERObjectIdentifier OID_PJ_DADOS_RESPONSAVEL =
            new DERObjectIdentifier("2.16.76.1.3.4");
    private static final DERObjectIdentifier OID_PF_ELEITORAL         =
            new DERObjectIdentifier("2.16.76.1.3.5");
    private static final DERObjectIdentifier OID_PF_CEI               =
            new DERObjectIdentifier("2.16.76.1.3.6");
    private static final DERObjectIdentifier OID_PJ_CEI               =
            new DERObjectIdentifier("2.16.76.1.3.7");
    private static final DERObjectIdentifier OID_PJ_NOME_EMPRESARIAL  =
            new DERObjectIdentifier("2.16.76.1.3.8");
```

sicid.util.CertificadoIcpBrasilParser.java

```
private static final DERObjectIdentifier OID_PF_RIC              =
        new DERObjectIdentifier("2.16.76.1.3.9");
private static final DERObjectIdentifier OID_PF_LOGIN            =
        new DERObjectIdentifier("1.3.6.1.4.1.311.20.2.3");
private static final DERObjectIdentifier OID_PJ_GUID_SERVER      =
        new DERObjectIdentifier("1.3.6.1.4.1.311.25.1");

/**
 * Retorna os atributos de um certificado digital, padrao ICP Brasil.
 * @param cert Objeto {@link X509Certificate} que representa o certificado.
 * @return Objeto {@link Map} contendo os atributos.
 * @throws Exception
 */
@SuppressWarnings("rawtypes")
public static Map<AtributoIcpBrasil, String> getAtributosIcpBrasil(
        X509Certificate cert) throws Exception {

    Map<AtributoIcpBrasil, String> atributos =
        new HashMap<AtributoIcpBrasil, String>();
    Collection subjAltNames =
        X509ExtensionUtil.getSubjectAlternativeNames(cert);
    for (Object obj : subjAltNames) {
        if (obj instanceof ArrayList) {
            Object key   = ((ArrayList) obj).get(0);
            Object value = ((ArrayList) obj).get(1);
            if (((Number) key).intValue() == 1) {
                String email =
                    extraiAtributo((String) value, -1, -1);
                atributos.put(AtributoIcpBrasil.EMAIL, email);
            }
            if (value instanceof ASN1Sequence) {
                ASN1Sequence seq = (ASN1Sequence) value;
                ASN1ObjectIdentifier oid =
                    (ASN1ObjectIdentifier) seq.getObjectAt(0);
                ASN1TaggedObject tagged =
                    (ASN1TaggedObject) seq.getObjectAt(1);
                String data = null;
                ASN1Primitive derObj = tagged.getObject();
                if (derObj instanceof DEROctetString) {
                    DEROctetString octet = (DEROctetString) derObj;
                    data = new String(octet.getOctets());
                } else if (derObj instanceof DERPrintableString) {
                    DERPrintableString octet = (DERPrintableString) derObj;
                    data = new String(octet.getOctets());
```

sicid.util.CertificadoIcpBrasilParser.java

```
} else if (derObj instanceof DERUTF8String) {
    DERUTF8String str = (DERUTF8String) derObj;
    data = str.getString();
}
if (data != null && !data.isEmpty()) {
    if (oid.equals(OID_PF_DADOS_TITULAR)
            || oid.equals(OID_PJ_DADOS_RESPONSAVEL)) {
        String nascimento = extraiAtributo(data, 0, 8);
        atributos.put(
            AtributoIcpBrasil.NASCIMENTO, nascimento);
        String cpf = extraiAtributo(data, 8, 19);
        atributos.put(AtributoIcpBrasil.CPF, cpf);
        String pisPasep = extraiAtributo(data, 19, 30);
        atributos.put(AtributoIcpBrasil.PISPASEP, pisPasep);
        String rg       = extraiAtributo(data, 30, 45);
        String rgOrgExp = extraiAtributo(data, 45, 49);
        String rgUF     = extraiAtributo(data, 49, 51);
        if ("".equals(rg)) {
            rgOrgExp = ""; rgUF = "";
        }
        atributos.put(AtributoIcpBrasil.RG, rg);
        atributos.put(
            AtributoIcpBrasil.RG_ORGEXPEDIDOR, rgOrgExp);
        atributos.put(AtributoIcpBrasil.RG_UF, rgUF);
    } else if (oid.equals(OID_PF_CEI)) {
        String cei = extraiAtributo(data, 0, 12);
        atributos.put(AtributoIcpBrasil.CEI, cei);
    } else if (oid.equals(OID_PF_ELEITORAL)) {
        String titulo    = extraiAtributo(data,  0, 12);
        String zona      = extraiAtributo(data, 12, 15);
        String secao     = extraiAtributo(data, 15, 19);
        String municipio = extraiAtributo(data, 19, 39);
        String uf        = extraiAtributo(data, 39, 41);
        if ("".equals(titulo)) {
            zona = ""; secao = ""; municipio = ""; uf = "";
        }
        atributos.put(AtributoIcpBrasil.TITULO, titulo);
        atributos.put(AtributoIcpBrasil.TITULO_ZONA, zona);
        atributos.put(AtributoIcpBrasil.TITULO_SECAO, secao);
        atributos.put(
            AtributoIcpBrasil.TITULO_MUNICIPIO, municipio);
        atributos.put(AtributoIcpBrasil.TITULO_UF, uf);
    } else if (oid.equals(OID_PJ_RESPONSAVEL)) {
        String nome = extraiAtributo(data, -1, -1);
        atributos.put(
            AtributoIcpBrasil.NOME_RESPONSAVEL, nome);
```

sicid.util.CertificadoIcpBrasilParser.java

```
                        } else if (oid.equals(OID_PJ_CNPJ)) {
                            String cnpj = extraiAtributo(data, -1, -1);
                            atributos.put(AtributoIcpBrasil.CNPJ, cnpj);
                        } else if (oid.equals(OID_PJ_CEI)) {
                            String cei = extraiAtributo(data, 0, 12);
                            atributos.put(AtributoIcpBrasil.CEI, cei);
                        } else if (oid.equals(OID_PJ_NOME_EMPRESARIAL)) {
                            String nomePJ = extraiAtributo(data, -1, -1);
                            atributos.put(AtributoIcpBrasil.NOME_PJ, nomePJ);
                        } else if (oid.equals(OID_PF_RIC)) {
                            String ric = extraiAtributo(data, -1, -1);
                            atributos.put(AtributoIcpBrasil.RIC, ric);
                        } else if (oid.equals(OID_PF_LOGIN)) {
                            String login = extraiAtributo(data, -1, -1);
                            atributos.put(AtributoIcpBrasil.LOGIN, login);
                        } else if (oid.equals(OID_PJ_GUID_SERVER)) {
                            String guid = extraiAtributo(data, -1, -1);
                            atributos.put(AtributoIcpBrasil.GUID_SERVER, guid);
                        }
                    }
                }
            }
        }
        /* se certificado for PF, nome esta' no subject CN, antes da separacao
         * por ':' (conforme ICP Brasil, CN = <nome>':'<cpf|ric>). */
        String nome = atributos.get(AtributoIcpBrasil.NOME_RESPONSAVEL);
        if (nome == null) {
            nome = extractNomeFromCN(cert.getSubjectX500Principal().getName());
            if (nome != null) {
                atributos.put(AtributoIcpBrasil.NOME_RESPONSAVEL, nome);
            }
        }
        return atributos;
    }

    /**
     * Extrai o nome do titular a partir do Common Name (CN), presente
     *   no Distinguished Name (DN) de um certificado padrao ICP Brasil.<br/>
     * Conforme especificacao, o campo CN e' composto de (nome):(documento).
     * @param dname Distinguished Name (DN) do certificado.
     * @return Nome extraido a partir do CN, ou null se nao encontrado.
     */
    public static String extractNomeFromCN(String dname) {
        /* se certificado for PF, nome esta' no subject CN, antes da separacao
         * por ':' (conforme ICP Brasil, CN = <nome>':'<cpf|ric>). */
        String nome = null;
        List<String> cnList = subjectParse(dname, "CN");
```

sicid.util.CertificadoIcpBrasilParser.java

```
        if (cnList.size() != 0) {
            String cn = cnList.get(0);
            if (cn != null) {
                int endIndex = cn.indexOf(":");
                if (endIndex >= 0) {
                    nome = cn.substring(0, endIndex);
                } else {
                    nome = cn;
                }
            }
        }
        return nome;
    }

    /**
     * Retorna os valores de um campo (field) de um subject
     *   (DN - Distinguished Name).
     * @param subject String representando um subject.
     * @param field Nome do campo a ser retornado (ex: "C", "O", "OU", etc.).
     * @return Valores atribuidos ao campo, ou vazio se nenhum.
     * @see <a href="http://www.x500standard.com/">X500 Standard</a>
     */
    public static List<String> subjectParse(String subject, String field) {
        List<String> subjList = new ArrayList<String>();
        if (subject != null && !"".equals(subject)) {
            String pattern = String.format("%s=", field);
            int beginIndex = 0;
            int endIndex = 0;
            do {
                beginIndex = subject.indexOf(pattern, beginIndex);
                if (beginIndex >= 0) {
                    beginIndex += pattern.length();
                    endIndex = subject.indexOf(",", beginIndex);
                    if (endIndex > - 0)
                        subjList.add(subject.substring(beginIndex, endIndex));
                    else
                        subjList.add(subject.substring(beginIndex));
                }
            } while (beginIndex >= 0);
        }
        return subjList;
    }
```

sicid.util.CertificadoIcpBrasilParser.java

```
    /* Extrai um atributo a partir de uma string.
     * @param value String contendo o atributo.
     * @param startIndex Indice de inicio do atributo na substring
     *   (se negativo, usa string inteira).
     * @param endIndex Indice de fim do atributo na substring.
     * @return String com o atributo, vazio se atributo estiver preenchido
     *   somente com zeros.
     */
    private static String extraiAtributo(
            String value, int startIndex, int endIndex) {

        String atributo = null;
        char zeroPattern[] = null;
        try {
            if (startIndex < 0) {
                atributo = value;
            } else {
                atributo = value.substring(startIndex, endIndex);
                zeroPattern = new char[endIndex - startIndex];
                Arrays.fill(zeroPattern, '0');
            }
            if (zeroPattern != null &&
                    (new String(zeroPattern)).equals(atributo)) {
                atributo = "";
            }
        } catch (Exception e) { /* retorna vazio apenas */ }
        if (atributo == null) { atributo = ""; }
        return atributo.trim();
    }
}
```

4.4 Cliente (Consumidor do *Web Service*) do SICid

sicid.ws.SICidClient.java

```
package sicid.ws;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URL;
import java.security.KeyPair;
import java.security.cert.X509Certificate;
import java.util.Map;
import java.util.Properties;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import sun.misc.BASE64Encoder;

import com.robsonmartins.fiap.tcc.util.CertificadoAssinador;
import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;

/**
 * Implementa um cliente para o Servico de Identificacao do Cidadao (SICid).
 *
 * <p>
 *   Descricao do arquivo properties, contendo opcoes de conexao:
 * <ul>
 * <li><em>sicid.url</em>: URL do Servico de Identificacao do Cidadao (SICid).
 * <li><em>sicid.namespace</em>: Namespace do Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.service</em>: Nome do Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.keyStore</em>: Nome do arquivo de keystore contendo o par de
 *   chaves do cliente para autenticar no Servico de Identificacao do Cidadao
 *   (SICid).
 * <li><em>sicid.storeType</em>: Tipo do arquivo de keystore (default: "JKS").
 * <li><em>sicid.storePass</em>: Senha do arquivo de keystore.
 * <li><em>sicid.keyAlias</em>: Alias do par de chaves a ser usado, dentro do
 *   arquivo de keystore.
 * <li><em>sicid.keyPass</em>: Senha do par de chaves dentro do
 *   arquivo de keystore.
 * </ul>
 * <p>
```

sicid.ws.SICidClient.java

```
 *
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class SICidClient {
    /* nomes das opcoes do cliente no arquivo properties */
    private final static String OPTION_SICID_URL          = "sicid.url"        ;
    private final static String OPTION_SICID_NAMESPACE    = "sicid.namespace"  ;
    private final static String OPTION_SICID_SERVICE      = "sicid.service"    ;
    private final static String OPTION_SICID_KEYSTORE     = "sicid.keyStore"   ;
    private final static String OPTION_SICID_KEYSTORE_TYPE = "sicid.storeType"  ;
    private final static String OPTION_SICID_KEYSTORE_PASS = "sicid.storePass"  ;
    private final static String OPTION_SICID_KEY_ALIAS    = "sicid.keyAlias"   ;
    private final static String OPTION_SICID_KEY_PASS     = "sicid.keyPass"    ;
    /* proxy para servico SICid */
    private ISICidService sicidService;
    private static Logger logger;
    private static boolean trace;

    /**
     * Cria uma nova instancia do cliente SICid.
     */
    public SICidClient() {
        logger = LogManager.getLogger(SICidClient.class);
        trace = logger.isTraceEnabled();
    }

    /**
     * Cria uma nova instancia do cliente SICid.
     * @param serviceURL URL do servico.
     * @param namespace Namespace do servico.
     * @param service Nome do servico.
     * @param keyStoreFile Nome do arquivo de keystore
     *   a ser usado para obter o par de chaves do cliente.
     * @param keyStoreType Tipo do arquivo de keystore (default: JKS).
     * @param keyAlias Alias do par de chaves dentro do arquivo de keystore.
     * @param keyStorePass Senha do arquivo de keystore.
     * @param keyPass Senha do par de chaves dentro do arquivo de keystore.
     * @throws Exception
     */
    public SICidClient(String serviceURL, String namespace, String service,
            String keyStoreFile, String keyStoreType, String keyAlias,
            String keyStorePass, String keyPass) throws Exception {

        this();
        connect(serviceURL, namespace, service, keyStoreFile, keyStoreType,
                keyAlias, keyStorePass, keyPass);
    }
```

sicid.ws.SICidClient.java

```
/**
 * Cria uma nova instancia do cliente SICid.
 * @param props Properties com as opcoes de conexao do cliente.
 * @throws Exception
 */
public SICidClient(Properties props) throws Exception {
    this();
    connect(props);
}

/**
 * Cria uma nova instancia do cliente SICid.
 * @param propsFileName Nome do arquivo properties
 *   com as opcoes de conexao do cliente.
 * @throws Exception
 */
public SICidClient(String propsFileName) throws Exception {
    this();
    connect(propsFileName);
}

/**
 * Conecta no Servico de Identificacao do Cidadao (SICid).
 * @param serviceURL URL do servico.
 * @param namespace Namespace do servico.
 * @param service Nome do servico.
 * @param keyStoreFile Nome do arquivo de keystore
 *   a ser usado para obter o par de chaves do cliente.
 * @param keyStoreType Tipo do arquivo de keystore (default: JKS).
 * @param keyAlias Alias do par de chaves dentro do arquivo de keystore.
 * @param keyStorePass Senha do arquivo de keystore.
 * @param keyPass Senha do par de chaves dentro do arquivo de keystore.
 * @throws Exception
 */
public void connect(String serviceURL, String namespace, String service,
        String keyStoreFile, String keyStoreType, String keyAlias,
        String keyStorePass, String keyPass) throws Exception {

    X509Certificate cert = null;
    KeyPair keys = null;
    String keyStoreFileFullPath = null;
    if (trace) {
        logger.trace(String.format(
                "Searching for keystore file %s", keyStoreFile));
    }
```

sicid.ws.SICidClient.java

```
        try {
            keyStoreFileFullPath = getKeyStoreFileFullPath(keyStoreFile);
            if (keyStoreFileFullPath == null) {
                logger.error(String.format(
                        "Keystore file %s not found", keyStoreFile));
                throw new FileNotFoundException(
                        String.format("File %s not found.", keyStoreFile));
            }
            if (trace) {
                logger.trace(String.format(
                        "Keystore file found in %s", keyStoreFileFullPath));
            }
            keys = CertificadoAssinador.getKeyPairFromFile(
                    keyStoreFileFullPath, keyStoreType,
                    keyAlias, keyStorePass, keyPass);
            cert = CertificadoAssinador.getCertFromFile(
                    keyStoreFileFullPath, keyStoreType,
                    keyAlias, keyStorePass);
            if (cert == null || keys == null) {
                throw new Exception(String.format(
                        "Error reading keystore file %s", keyStoreFileFullPath));
            }
        } catch (Exception e) {
            logger.error(String.format(
                    "Error reading keystore file %s", keyStoreFile), e);
            Exception exception = new Exception(String.format(
                    "Error reading keystore file %s", keyStoreFile));
            exception.initCause(e);
            throw exception;
        }
        if (trace) {
            logger.trace(String.format(
                    "Connecting with SICid" +
                    " (url=%s namespace=%s service=%s cert:DN=%s)",
                    serviceURL, namespace, service,
                    cert.getSubjectX500Principal().getName()));
        }
        try {
            URL wsdlUrl = new URL(serviceURL + "?wsdl");
            QName qnameService  = new QName(namespace, service);
            /* configura autenticador HTTP/BASIC para ler o WSDL */
            SICidBasicAuthenticator authenticator =
                new SICidBasicAuthenticator(cert, keys);
            Authenticator.setDefault(authenticator);
            Service wsService = Service.create(wsdlUrl, qnameService);
            sicidService = wsService.getPort(ISICidService.class);
```

sicid.ws.SICidClient.java

```
            /* configura parametros de autenticacao para o proxy do servico */
            Map<String, Object> reqContext =
                ((BindingProvider) sicidService).getRequestContext();
            reqContext.put(BindingProvider.USERNAME_PROPERTY,
                    authenticator.getUsername());
            reqContext.put(BindingProvider.PASSWORD_PROPERTY,
                    authenticator.getPassword());
            if (trace) {
                logger.trace("Connected with SICid service");
            }
        } catch (Exception e) {
            logger.error("Error connecting with SICid service", e);
            Exception exception =
                new Exception("Error connecting with SICid service");
            exception.initCause(e);
            throw exception;
        }
    }

    /**
     * Conecta no Servico de Identificacao do Cidadao (SICid).
     * @param props Properties com as opcoes de conexao do cliente.
     * @throws Exception
     */
    public void connect(Properties props) throws Exception {
        String serviceURL   = props.getProperty(OPTION_SICID_URL          );
        String namespace    = props.getProperty(OPTION_SICID_NAMESPACE    );
        String service      = props.getProperty(OPTION_SICID_SERVICE      );
        String keyStoreFile = props.getProperty(OPTION_SICID_KEYSTORE     );
        String keyStoreType = props.getProperty(OPTION_SICID_KEYSTORE_TYPE);
        String keyStorePass = props.getProperty(OPTION_SICID_KEYSTORE_PASS);
        String keyAlias     = props.getProperty(OPTION_SICID_KEY_ALIAS    );
        String keyPass      = props.getProperty(OPTION_SICID_KEY_PASS     );
        connect(serviceURL, namespace, service, keyStoreFile,
                keyStoreType, keyAlias,    keyStorePass, keyPass);
    }

    /**
     * Conecta no Servico de Identificacao do Cidadao (SICid).
     * @param propsFileName Nome do arquivo properties
     *   com as opcoes de conexao do cliente.
     * @throws Exception
     */
    public void connect(String propsFileName) throws Exception {
        Properties props = getOptionsFromFile(propsFileName);
        connect(props);
    }
```

sicid.ws.SICidClient.java

```
/**
 * Retorna as opcoes do cliente SICid definidas em arquivo properties.
 * @param propsFileName Nome do arquivo properties
 *   com as opcoes de conexao do cliente.
 * @return Opcoes presentes no arquivo.
 * @throws Exception
 */
public Properties getOptionsFromFile(String propsFileName) throws Exception {
    Properties props = new Properties();
    InputStream inputStream = null;
    String confDir = null;
    /* tenta obter path "conf" do JBoss */
    confDir = System.getProperty("jboss.server.config.url");
    if (confDir == null) {
        /* tenta obter path "serverhome"/conf do JBoss */
        confDir = System.getProperty("jboss.server.home.url");
        if (confDir != null) { confDir += File.separatorChar + "conf"; }
    }
    if (confDir != null) {
        /* tenta obter arquivo no path especificado */
        try {
            inputStream =
                this.getClass().getClassLoader().getResourceAsStream(
                        String.format("%s%s%s", confDir,
                                        File.separator, propsFileName));
        } catch (Exception e) { }
    }
    if (inputStream == null) {
        /* se nao achou no dir JBoss, tenta obter arquivo via classloader */
        try {
            inputStream = this.getClass().getClassLoader()
                .getResourceAsStream(propsFileName);
        } catch (Exception e) {    }
    }
    if (inputStream == null) {
        /* se nao achou, tenta obter arquivo diretamente */
        try {
            inputStream = new FileInputStream(propsFileName);
        } catch (Exception e) {    }
    }
    if (inputStream == null) {
        throw new FileNotFoundException(
                String.format("File %s not found.", propsFileName));
    }
    props.load(inputStream);
    return props;
}
```

sicid.ws.SICidClient.java

```
/**
 * Retorna o proxy para o servico remoto SICid.
 * @return Proxy para o servico SICid.
 */
public ISICidService getService() {
    return sicidService;
}

/* Retorna o caminho completo do arquivo de KeyStore.
 * @param keyStoreFile Nome do arquivo de KeyStore.
 * @return Caminho completo do arquivo, ou null se nao encontrado.
 */
private String getKeyStoreFileFullPath(String keyStoreFile) {
    String confDir = null;
    InputStream inputStream = null;
    String ksFileRealPath = null;
    /* tenta obter path "conf" do JBoss */
    confDir = System.getProperty("jboss.server.config.url");
    if (confDir == null) {
        /* tenta obter path "serverhome"/conf do JBoss */
        confDir = System.getProperty("jboss.server.home.url");
        if (confDir != null) { confDir += File.separatorChar + "conf"; }
    }
    if (confDir != null) {
        /* tenta obter arquivo no path especificado */
        try {
            ksFileRealPath =
                this.getClass().getClassLoader().getResource(
                        String.format("%s%s%s", confDir,
                                File.separator, keyStoreFile)).getFile();
            inputStream = new FileInputStream(ksFileRealPath);
        } catch (Exception e) { }
    }
    if (inputStream == null) {
        /* se nao achou no dir JBoss, tenta obter arquivo via classloader */
        try {
            ksFileRealPath =
                this.getClass().getClassLoader().getResource(
                        keyStoreFile).getFile();
            inputStream = new FileInputStream(ksFileRealPath);
        } catch (Exception e) {    }
    }
    if (inputStream == null) {
        /* se nao achou, tenta obter arquivo diretamente */
        try {
            inputStream = new FileInputStream(keyStoreFile);
            if (inputStream != null) { ksFileRealPath = keyStoreFile; }
```

sicid.ws.SICidClient.java

```
            } catch (Exception e) {     }
        }
        if (inputStream == null) { ksFileRealPath = null; }
        try { inputStream.close(); } catch (Exception e) { }
        return ksFileRealPath;
    }

    /* Implementa um autenticador HTTP/BASIC para o cliente do
     *   Servico de Identificacao do Cidadao (SICid).
     */
    private class SICidBasicAuthenticator extends Authenticator {
        private String username;
        private String password;

        /* Cria uma nova instancia do autenticador.
         * @param cert Certificado do cliente.
         * @param keys Par de chaves do cliente.
         * @throws Exception
         */
        public SICidBasicAuthenticator(X509Certificate cert,
                KeyPair keys) throws Exception {
            /* username recebe o conteudo do certificado,
             * em formato Base64 */
            this.username = CertificadoSerializador.certToStr(cert);
            /* password recebe a assinatura do conteudo do certificado,
             * em formato Base64 */
            this.password = new BASE64Encoder().encode(
                    CertificadoAssinador.sign(
                        keys.getPrivate(), cert.getEncoded()))
                            .replaceAll("\\s+","");
            if (trace) {
                logger.trace(
                    String.format("SICid Basic Authenticator: uname=%s",
                                  this.username));
                logger.trace(
                    String.format("SICid Basic Authenticator: pass=%s",
                                  this.password));
            }
        }

        /* Retorna o valor do campo Username.
         * @return Valor de Username.
         */
        public String getUsername() {
            return username;
        }
```

sicid.ws.SICidClient.java

```
        /* Retorna o valor do campo Password.
         * @return Valor de Password.
         */
        public String getPassword() {
            return password;
        }

        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password.toCharArray());
        }
    }
}
```

4.5 Aplicação *Web* para Administração do SICid

sicid.web.controller.SICidMB.java

```
package sicid.web.controller;

import java.io.Serializable;
import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;

import com.robsonmartins.fiap.tcc.util.FacesUtil;
import sicid.bean.Usuario;
import sicid.model.ISICidEngine;

/**
 * Managed Bean responsavel por fornecer o acesso 'as funcionalidades de
 *   administracao do Servico de Identificacao do Cidadao (SICid)
 *   da camada de visualizacao do JSF.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@ManagedBean(name="sicid")
@SessionScoped
@SuppressWarnings("serial")
public class SICidMB implements Serializable {
    /* motor do SICid */
    @EJB
    private ISICidEngine sicidEngine;
    /* controller do cadastro de certificados confiaveis */
    private CertificadoConfiavelMB certMB;
    /* controller do cadastro de aplicacoes confiaveis */
    private ConsumidorConfiavelMB appMB;
    /* controller do cadastro de usuarios */
    private UsuarioMB usuarioMB;
    /* controller do cadastro nacional do cidadao */
    private CidadaoMB cidadaoMB;

    /* inicializa ManagedBeans */
    @PostConstruct
    protected void init() {
        certMB      = new CertificadoConfiavelMB(sicidEngine);
        appMB       = new ConsumidorConfiavelMB (sicidEngine);
        usuarioMB   = new UsuarioMB             (sicidEngine);
        cidadaoMB   = new CidadaoMB             (sicidEngine);
    }
```

sicid.web.controller.SICidMB.java

```
/**
 * Action para realizar logoff.
 * @return Action redirecionada apos logoff.
 */
public String logoff() {
    HttpServletRequest request = FacesUtil.getRequest();
    try {
        request.logout();
    } catch (ServletException e) { }
    request.getSession().invalidate();
    return "cidadao";
}

/**
 * Obtem o Usuario logado.
 * @return Objeto que representa o Usuario logado.
 */
public Usuario getUsuarioLogin() {
    return usuarioMB.getUsuarioLogin();
}

/**
 * Retorna o nome completo do usuario logado.
 * @return Nome do usuario logado.
 */
public String getLoginName() {
    Usuario usuario = getUsuarioLogin();
    return (usuario != null) ? usuario.getName() : "";
}

/**
 * Retorna o controller que gerencia o cadastro de
 *   Certificados Confiaveis.
 * @return Controller do cadastro de Certificados Confiaveis.
 */
public CertificadoConfiavelMB getCert() {
    return certMB;
}

/**
 * Configura o controller que gerencia o cadastro de
 *   Certificados Confiaveis.
 * @param cert Controller do cadastro de Certificados Confiaveis.
 */
public void setCert(CertificadoConfiavelMB cert) {
    this.certMB = cert;
}
```

sicid.web.controller.SICidMB.java

```
/**
 * Retorna o controller que gerencia o cadastro de
 *   Aplicacoes Confiaveis.
 * @return Controller do cadastro de Aplicacoes Confiaveis.
 */
public ConsumidorConfiavelMB getApp() {
    return appMB;
}

/**
 * Configura o controller que gerencia o cadastro de
 *   Aplicacoes Confiaveis.
 * @param app Controller do cadastro de Aplicacoes Confiaveis.
 */
public void setApp(ConsumidorConfiavelMB app) {
    this.appMB = app;
}

/**
 * Retorna o controller que gerencia o cadastro de Usuarios.
 * @return Controller do cadastro de Usuarios.
 */
public UsuarioMB getUsuario() {
    return usuarioMB;
}

/**
 * Configura o controller que gerencia o cadastro de Usuarios.
 * @param usuario Controller do cadastro de Usuarios.
 */
public void setUsuario(UsuarioMB usuario) {
    this.usuarioMB = usuario;
}

/**
 * Retorna o controller que gerencia o cadastro de Cidadaos.
 * @return Controller do cadastro de Cidadaos.
 */
public CidadaoMB getCidadao() {
    return cidadaoMB;
}
```

sicid.web.controller.SICidMB.java

```
    /**
     * Configura o controller que gerencia o cadastro de Cidadaos.
     * @param cidadao Controller do cadastro de Cidadaos.
     */
    public void setCidadao(CidadaoMB cidadao) {
        this.cidadaoMB = cidadao;
    }
}
```

sicid.web.controller.AbstractSICidMB.java

```
package sicid.web.controller;

import java.io.Serializable;

import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;
import com.robsonmartins.fiap.tcc.util.FacesUtil;
import sicid.bean.Usuario;
import sicid.model.ISICidEngine;

/**
 * Classe base abstrata, responsavel por fornecer o acesso 'as funcionalidades
 *   de administracao do Servico de Identificacao do Cidadao (SICid)
 *   'a camada de visualizacao do JSF.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@SuppressWarnings("serial")
public abstract class AbstractSICidMB implements Serializable {
    /* acao selecionada na camada de visualizacao */
    protected String actionSelecionada;
    /* motor do SICid */
    protected ISICidEngine sicidEngine;
    /* armazena usuario logado */
    protected String usernameLogado;
    protected Usuario usuarioLogado;

    /**
     * Cria uma nova instancia do controller.
     * @param engine "Motor" do SICid (camada model).
     */
    public AbstractSICidMB(ISICidEngine engine) {
        sicidEngine = engine;
    }

    /**
     * Obtem o Usuario logado.
     * @return Objeto que representa o Usuario logado.
     */
    protected Usuario getUsuarioLogin() {
        String username = FacesUtil.getUserPrincipalName();
        if (username != null && !username.equals(usernameLogado)) {
            usernameLogado = username;
            String dname = null;
            try {
                dname = CertificadoSerializador.getDNByCertStr(usernameLogado);
                usuarioLogado = sicidEngine.localizarUsuario(dname);
            } catch (Exception e) { }
```

sicid.web.controller.AbstractSICidMB.java

```
            if (usuarioLogado == null) {
                usuarioLogado = new Usuario();
                usuarioLogado.setUsername("admin");
                usuarioLogado.setName("Administrador");
                usuarioLogado.setRole("admin");
            }
        } else if (username == null) {
            usuarioLogado = null;
        }
        return usuarioLogado;
    }

    /**
     * Retorna a selecao de action.
     * @return Acao selecionada.
     */
    public String getActionSelecionada() {
        return actionSelecionada;
    }

    /**
     * Armazena a selecao de action.
     * @param actionSelecionada Acao selecionada.
     */
    public void setActionSelecionada(String actionSelecionada) {
        this.actionSelecionada = actionSelecionada;
    }
}
```

sicid.web.controller.CidadaoMB.java

```
package sicid.web.controller;

import java.io.InputStream;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.List;
import javax.faces.application.FacesMessage;
import org.primefaces.component.fileupload.FileUpload;
import org.primefaces.event.FileUploadEvent;

import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;
import com.robsonmartins.fiap.tcc.util.FacesUtil;
import sicid.bean.Cidadao;
import sicid.bean.DocumentoRG;
import sicid.bean.DocumentoTitulo;
import sicid.model.ISICidEngine;

/**
 * Classe responsavel por fornecer o acesso 'as funcionalidades do
 *   Cadastro Nacional do Cidadao, do Servico de Identificacao
 *   do Cidadao (SICid) 'a camada de visualizacao do JSF.
 * @author Robson Martins (robson@robsonmartins.com)
 */
@SuppressWarnings("serial")
public class CidadaoMB extends AbstractSICidMB {
    /* objeto cidadao selecionado na camada de visualizacao */
    private Cidadao cidadaoSelecionado;
    /* objeto que representa um novo cidadao */
    private Cidadao newCidadao;
    /* lista de cidadaos cadastrados no SICid */
    private List<Cidadao> cidadaos;

    /**
     * Cria uma nova instancia do controller.
     * @param engine "Motor" do SICid (camada model).
     */
    public CidadaoMB(ISICidEngine engine) {
        super(engine);
        cidadaos = new ArrayList<Cidadao>();
        atualizarListaCidadaos();
        criaNovoCidadao();
    }

    /**
     * Retorna uma lista de cidadaos cadastrados.
     * @return Lista de cidadaos.
     */
```

sicid.web.controller.CidadaoMB.java

```
public List<Cidadao> getListarCidadaos() {
    return cidadaos;
}

/**
 * Action para redirecionar ao cadastro de cidadaos.
 * @return String com o nome do destino (target) do redirecionamento
 * da action.
 */
public String goToCidadao() {
    cidadaoSelecionado = null;
    atualizarListaCidadaos();
    return "cidadao";
}

/**
 * Action para redirecionar 'a pagina de novo cidadao.
 * @return String com o nome do destino (target) do redirecionamento
 * da action.
 */
public String goToNovoCidadao() {
    cidadaoSelecionado = null;
    criaNovoCidadao();
    atualizarListaCidadaos();
    return "newcidadao";
}

/**
 * Action para redirecionar 'a pagina de editar cidadao.
 * @return String com o nome do destino (target) do redirecionamento
 * da action.
 */
public String goToEditarCidadao() {
    newCidadao = cidadaoSelecionado.clone();
    if (newCidadao == null) { criaNovoCidadao(); }
    atualizarListaCidadaos();
    return "newcidadao";
}

/**
 * Event Handler do componente {@link FileUpload}, para adicionar um
 *   certificado de cidadao ao cadastro.
 * @param event Objeto Event do componente FileUpload (Primefaces).
 */
public void uploadCertCidadao(FileUploadEvent event) {
    try {
        InputStream istream = event.getFile().getInputstream();
```

sicid.web.controller.CidadaoMB.java

```
            X509Certificate cert =
                CertificadoSerializador.loadCertFromStream(istream);
            String content =
                CertificadoSerializador.certToStr(cert);
            newCidadao = sicidEngine.getCidadaoInfoFromCert(content);
            if (newCidadao == null) {
                criaNovoCidadao();
            }
        } catch (Exception e) {
            FacesUtil.addFacesMessage(
                "Erro ao obter informa\u00E7\u00F5es do certificado.",
                e.getLocalizedMessage(), FacesMessage.SEVERITY_ERROR);
        }
    }

    /**
     * Action para adicionar um cidadao ao cadastro.
     */
    public String adicionarCidadao() {
        try {
            if (newCidadao == null) {
                throw new NullPointerException(
                    "Novo cidad\u00E3o inv\u00E1lido.");
            }
            sicidEngine.adicionarCidadao(newCidadao);
            FacesUtil.addFacesMessage("Cidad\u00E3o adicionado com sucesso.",
                    null, FacesMessage.SEVERITY_INFO);
            criaNovoCidadao();
            atualizarListaCidadaos();
            return goToCidadao();
        } catch (Exception e) {
            FacesUtil.addFacesMessage("Erro ao adicionar o cidad\u00E3o.",
                    e.getLocalizedMessage(), FacesMessage.SEVERITY_ERROR);
            return null;
        }
    }

    /**
     * Action para remover um cidadao do cadastro.
     */
    public void removerCidadao() {
        try {
            sicidEngine.removerCidadao(cidadaoSelecionado.getDname());
            FacesUtil.addFacesMessage(
                "Cidad\u00E3o removido com sucesso.", null,
                FacesMessage.SEVERITY_INFO);
            cidadaoSelecionado = null;
```

sicid.web.controller.CidadaoMB.java

```
            atualizarListaCidadaos();
        } catch (Exception e) {
            FacesUtil.addFacesMessage(
                "Erro ao remover o cidad\u00E3o.", e.getLocalizedMessage(),
                FacesMessage.SEVERITY_ERROR);
        }
    }

    /**
     * Retorna a selecao de cidadao.
     * @return Cidadao selecionado.
     */
    public Cidadao getCidadaoSelecionado() {
        return cidadaoSelecionado;
    }

    /**
     * Armazena a selecao de cidadao.
     * @param cidadaoSelecionado Cidadao selecionado.
     */
    public void setCidadaoSelecionado(Cidadao cidadaoSelecionado) {
        this.cidadaoSelecionado = cidadaoSelecionado;
    }

    /**
     * Retorna o novo cidadao.
     * @return Novo cidadao.
     */
    public Cidadao getNewCidadao() {
        return newCidadao;
    }

    /**
     * Armazena o novo cidadao.
     * @param newCidadao Novo cidadao.
     */
    public void setNewCidadao(Cidadao newCidadao) {
        this.newCidadao = newCidadao;
    }
```

sicid.web.controller.CidadaoMB.java

```
    /**
     * Cria um objeto representando um novo cidadao.
     */
    public void criaNovoCidadao() {
        this.newCidadao = new Cidadao();
        this.newCidadao.setRg(new DocumentoRG());
        this.newCidadao.setTitulo(new DocumentoTitulo());
    }

    /* Atualiza lista de cidadaos. */
    private void atualizarListaCidadaos() {
        cidadaos.clear();
        cidadaos.addAll(sicidEngine.listarCidadaos());
    }
}
```

com.robsonmartins.fiap.tcc.util.FacesUtil.java

```
package com.robsonmartins.fiap.tcc.util;

import java.security.Principal;
import javax.faces.application.FacesMessage;
import javax.faces.application.FacesMessage.Severity;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;

/**
 * Encapsula funcionalidades uteis no contexto de JSF (Java Server Faces).
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class FacesUtil {
    /**
     * Obtem o obj da requisicao HTTP a partir do contexto do servlet.
     * @return Objeto da requisicao HTTP.
     */
    public static HttpServletRequest getRequest() {
        HttpServletRequest request =
            (HttpServletRequest) FacesContext.getCurrentInstance()
                .getExternalContext().getRequest();
        return request;
    }

    /**
     * Retorna o username do Usuario logado na sessao HTTP.
     * @return Username do usuario logado na sessao corrente.
     */
    public static String getUserPrincipalName() {
        Principal principal = null;
        HttpServletRequest request = getRequest();
        if (request != null) { principal = request.getUserPrincipal(); }
        return (principal != null) ? principal.getName() : null;
    }

    /**
     * Adiciona uma mensagem no contexto do JSF.
     * @param summary Texto de sumario da mensagem.
     * @param detail Texto de detalhe da mensagem.
     * @param severity Severidade da mensagem.
     */
    public static void addFacesMessage(String summary, String detail,
                                       Severity severity) {
        FacesMessage facesMessage = new FacesMessage(severity, summary, detail);
        FacesContext.getCurrentInstance().addMessage(null, facesMessage);
    }
}
```

sicid/admin/cidadao.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition template="../template/main.xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui"
    xmlns:f="http://java.sun.com/jsf/core">

  <ui:define name="headTitle">
    <title>Cadastro Nacional do Cidadão - SICid Admin</title>
  </ui:define>
  <ui:define name="title">
    <h1>Cadastro do Cidadão</h1>
  </ui:define>
  <ui:define name="content">
    <h:form id="formCidadao" method="post">
      <p:growl id="messages" showDetail="false"/>
      <p:panel id="btnPanel"
        style="text-align:center;border:1px solid silver;margin:0px;padding:0px">
        <h:panelGrid columns="5" cellpadding="0">
          <p:commandButton id="addButton" icon="ui-icon-plus"
            value="Cadastrar Cidadão" title="Adicionar Cidadão ao Cadastro"
            ajax="false" action="#{sicid.cidadao.goToNovoCidadao}" />
          <p:spacer width="4px" />
          <p:commandButton id="renButton" icon="ui-icon-pencil"
            value="Editar Cidadão" title="Editar Cadastro do Cidadão"
            ajax="false" action="#{sicid.cidadao.goToEditarCidadao}"
            disabled="#{empty sicid.cidadao.cidadaoSelecionado}" />
          <p:spacer width="4px" />
          <p:commandButton id="delButton" update=":formCidadao:dlgConfirm,
            :formCidadao:messages,:formCidadao:statusDialog"
            icon="ui-icon-cancel" value="Remover Cidadão" title="Remover Cidadão"
            oncomplete="dlgConfirm.show();"
            disabled="#{empty sicid.cidadao.cidadaoSelecionado}" />
        </h:panelGrid>
      </p:panel>
      <p:spacer width="100%" height="4px" />
      <p:dataTable var="cidadao" value="#{sicid.cidadao.listarCidadaos}"
        id="cidadaoList" sortBy="#{cidadao.nome}"
        editable="false" emptyMessage="Nenhum cidadão cadastrado."
        scrollable="true" scrollHeight="350" style="text-align:left"
        selection="#{sicid.cidadao.cidadaoSelecionado}" selectionMode="single"
        rowKey="#{cidadao.dname}">
        <p:ajax event="rowSelect" update=":formCidadao:btnPanel" />
```

sicid/admin/cidadao.xhtml

```
            <p:column headerText="Nome" width="300"
              sortBy="#{cidadao.nome}">
              <h:outputText value="#{cidadao.nome}" />
            </p:column>
            <p:column headerText="Nascimento" width="80"
               sortBy="#{cidadao.dataNascimento}">
               <h:outputText value="#{cidadao.dataNascimento}">
                 <f:convertDateTime pattern="dd/MM/yyyy" />
               </h:outputText>
            </p:column>
            <p:column headerText="Nome da Mãe" width="300"
               sortBy="#{cidadao.nomeMae}">
               <h:outputText value="#{cidadao.nomeMae}" />
            </p:column>
            <p:column headerText="RIC" width="95"
                sortBy="#{cidadao.ric}">
                <h:outputText value="#{cidadao.ric}" />
            </p:column>
            <p:column headerText="CPF" width="95"
                sortBy="#{cidadao.cpf}">
                <h:outputText value="#{cidadao.cpf}" />
            </p:column>
        </p:dataTable>
        <p:confirmDialog
          showEffect="bounce" hideEffect="explode" modal="true" id="dlgConfirm"
          header="Remover Cidadão" severity="alert" widgetVar="dlgConfirm"
          closable="false"
          message="Tem certeza que deseja remover o cidadão selecionado?">
          <p align="center">
          <p:commandButton value="Sim"
            onclick="dlgConfirm.hide();statusDialog.show();"
            update=":formCidadao:cidadaoList,:formCidadao:btnPanel"
            action="#{sicid.cidadao.removerCidadao}"
            onsuccess="statusDialog.hide();" />  
          <p:commandButton value="Não" onclick="dlgConfirm.hide()" type="button"/>
        </p>
      </p:confirmDialog>
      <p:dialog id="statusDialog" modal="true" widgetVar="statusDialog"
        header="Processando Cadastro. Aguarde..."
        draggable="false" closable="false" resizable="false" dynamic="true">
        <p:graphicImage value="../images/ajaxloadingbar.gif" />
      </p:dialog>
    </h:form>
  </ui:define>
</ui:composition>
```

sicid/template/main.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui"
  xmlns:f="http://java.sun.com/jsf/core">

<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <ui:insert name="headTitle"/>
  <link rel="stylesheet" type="text/css" href="../style/style.css" />
  <link rel="shortcut icon" href="../favicon.ico" />
</h:head>

<h:body>
  <f:view contentType="text/html" encoding="UTF-8">
  <p:growl id="messages" showDetail="false"/>
  <div align="center">
    <div style="margin-top:4px">
      <div class="ui-widget-header"
        style="margin:0px;padding:0px;width:100%;text-align:left;">
        <h:form id="formHeader" method="post" prependId="false">
          <h:panelGrid columns="3" cellpadding="2">
            <p:graphicImage value="../images/logo.png" />
            <p:spacer width="50px" height="10px" />
            <ui:insert name="title"/>
          </h:panelGrid>
          <p:menubar>
            <p:menuitem value="Cadastro Nacional do Cidadão"
              action="#{sicid.cidadao.goToCidadao}" icon="ui-icon-person" />
            <p:menuitem value="Certificados Confiáveis"
              action="#{sicid.cert.goToTrustedCert}" icon="ui-icon-locked" />
            <p:menuitem value="Aplicações Confiáveis"
              action="#{sicid.app.goToTrustedApps}" icon="ui-icon-newwin" />
            <p:menuitem value="Administradores"
              action="#{sicid.usuario.goToTrustedUsers}" icon="ui-icon-key" />
            <p:menuitem value="Logoff: #{sicid.loginName}"
              action="#{sicid.logoff}" icon="ui-icon-transferthick-e-w"
              rendered="#{!empty sicid.loginName}" />
          </p:menubar>
        </h:form>
      </div>
    </div>
    <div style="margin-top:4px">
      <div style="border:1px solid silver;margin:0px;padding:0px;width:100%">
```

sicid/template/main.xhtml

```
        <ui:insert name="content" />
      </div>
      <div align="right" class="ui-widget-header"
           style="margin:0px;padding:10px;width:98%">
        SICid Admin - (2012) Robson Martins / (TCC) FIAP 16SCJ
      </div>
      <p:messages id="dmessages" showDetail="true" autoUpdate="true"
                  severity="error"/>
    </div>
  </div>
</f:view>
</h:body>
</html>
```

4.6 Configuração do Banco Seguro para usar o SICid

bancoseguro/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <display-name>BancoSeguro</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>org.primefaces.webapp.filter.FileUploadFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
  </filter-mapping>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>All resources</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/jaas/login.jsf</form-login-page>
      <form-error-page>/jaas/error.jsf</form-error-page>
    </form-login-config>
  </login-config>
```

bancoseguro/web.xml

```
<error-page>
  <error-code>403</error-code>
  <location>/jaas/autherror.jsf</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/jaas/notfound.jsf</location>
</error-page>
<security-role>
  <description>Gerente</description>
  <role-name>gerente</role-name>
</security-role>
<security-role>
  <description>Cliente</description>
  <role-name>cliente</role-name>
</security-role>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Gerente</web-resource-name>
    <description>Recursos de Gerente</description>
    <url-pattern>/admin/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>Permissoes para Gerente</description>
    <role-name>gerente</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Cliente</web-resource-name>
    <description>Recursos de Cliente</description>
    <url-pattern>/user/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>Permissoes para Cliente</description>
    <role-name>cliente</role-name>
    <role-name>gerente</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
  <web-resource-name>Acesso Publico</web-resource-name>
  <description>Recursos Liberados</description>
```

bancoseguro/web.xml

```
    <url-pattern>/*</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>PUT</http-method>
    <http-method>HEAD</http-method>
    <http-method>OPTIONS</http-method>
    <http-method>TRACE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    </web-resource-collection>
  </security-constraint>
</web-app>
```

bancoseguro/jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
           http://www.jboss.org/j2ee/schema/jboss-web_6_0.xsd"
           version="6.0">
  <security-domain>java:/jaas/BancoSeguro</security-domain>
</jboss-web>
```

bancoseguro/jaas/login.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">
<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Autenticação - Banco Seguro</title>
  <link rel="stylesheet" type="text/css" href="../style/style.css"
        media="screen" />
  <link rel="shortcut icon" href="../favicon.ico" />
</h:head>

<h:body>
  <f:view contentType="text/html">
  <div align="center">
    <div style="margin-top:20px">
      <div class="ui-widget-header"
           style="margin:0px;padding:0px;width:100%;text-align:left;">
        <h:form id="formHeader" method="post" prependId="false">
          <h:panelGrid columns="3" cellpadding="2">
            <p:graphicImage value="../images/logo.png" />
            <p:spacer width="50px" height="10px" />
            <h1>Login</h1>
          </h:panelGrid>
        </h:form>
      </div>
    </div>
    <div style="margin-top:4px">
      <div style="border:1px solid silver;margin:0px;padding:0px;width:100%">
        <form id="formLogin" name="formLogin"
              action="j_security_check" method="post">
          <p:panel style="width:445px;border:none" id="panelLogin">
            <object classid="java:SICidApplet.class"
                    type="application/x-java-applet"
                    archive=
                     "https://tcc.fiap.robsonmartins.com/sicid/ui/SICidApplet.jar"
                    height="250" width="400" >
              <param name="archive"
                     value=
                      "https://tcc.fiap.robsonmartins.com/sicid/ui/SICidApplet.jar"
              />
              <param name="form" value="formLogin" />
              <param name="username" value="j_username" />
```

bancoseguro/jaas/login.xhtml

```
            <param name="password" value="j_password" />
            <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
                    height="250" width="400" >
              <param name="code" value="SICidApplet" />
              <param name="archive"
                     value=
                   "https://tcc.fiap.robsonmartins.com/sicid/ui/SICidApplet.jar"
              />
              <param name="form" value="formLogin" />
              <param name="username" value="j_username" />
              <param name="password" value="j_password" />
              O plugin Java é necessário para executar esta aplicação.
            </object>
          </object>
          <input type="text" name="j_username" id="j_username"
                 style="visibility:hidden" />
          <input type="text" name="j_password" id="j_password"
                 style="visibility:hidden" />
        </p:panel>
      </form>
    </div>
    <div align="right" class="ui-widget-header"
         style="margin:0px;padding:10px;width:98%">
      Banco Seguro - (2012) Robson Martins / (TCC) FIAP 16SCJ
    </div>
    <div align="center" style="margin-top:10px">
      <ui:insert name="loginMessage" />
    </div>
  </div>
</div>
</f:view>
</h:body>

</html>
```

Banco Seguro: <jboss-server-home>/conf/login-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
  <application-policy name="BancoSeguro">
    <authentication>
      <login-module
           code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
           flag="required">
        <module-option name="dsJndiName">java:/BancoSeguroDS</module-option>
        <module-option name="principalsQuery">
          SELECT cpf as PrincipalID FROM Usuario WHERE dname=?
        </module-option>
        <module-option name="rolesQuery">
          SELECT role as Role, 'Roles' FROM Usuario WHERE cpf=?
        </module-option>
        <module-option name="fullPrincipalsQuery">
          SELECT * FROM Usuario
        </module-option>
        <module-option name="principalIdForEmpty">gerente</module-option>
        <module-option name="roleForEmpty">gerente</module-option>
        <module-option name="sicid.clientProps">
          bancoseguro.properties
        </module-option>
      </login-module>
    </authentication>
  </application-policy>

  <application-policy name="BancoSeguroService">
    <authentication>
      <login-module
           code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
           flag="required">
        <module-option name="dsJndiName">java:/BancoSeguroDS</module-option>
        <module-option name="principalsQuery">
          SELECT id as PrincipalID from ConsumidorConfiavel WHERE dname=?
        </module-option>
        <module-option name="rolesQuery">
          SELECT role as Role, 'Roles' from ConsumidorConfiavel WHERE id=?
        </module-option>
        <module-option name="fullPrincipalsQuery">
          SELECT * FROM ConsumidorConfiavel
        </module-option>
        <module-option name="principalIdForEmpty">wsuser</module-option>
        <module-option name="roleForEmpty">wsuser</module-option>
        <module-option name="sicid.clientProps">
          bancoseguro.properties
        </module-option>
      </login-module>
```

Banco Seguro: <jboss-server-home>/conf/login-config.xml

```
    </authentication>
  </application-policy>
</policy>
```

Banco Seguro: <jboss-server-home>/conf/bancoseguro.properties

```
sicid.url=https://tcc.fiap.robsonmartins.com/sicid/service/sicid
sicid.namespace=http://ws.sicid/
sicid.service=SICidService
sicid.keyStore=props/bancoseguro.keystore
sicid.storeType=JKS
sicid.storePass=banco
sicid.keyAlias=banco
sicid.keyPass=banco
```

Banco Seguro: <jboss-server-home>/deploy/bancoseguro-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC
        "-//JBoss//DTD Web Application 5.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>BancoSeguroDS</jndi-name>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <connection-url>
      jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}bancoseguroDB
    </connection-url>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

bancoseguro/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

  <persistence-unit name="bancoseguro">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:/BancoSeguroDS</jta-data-source>
    <class>banco.bean.ConsumidorConfiavel</class>
    <class>banco.bean.Usuario</class>
    <class>banco.bean.Conta</class>
    <class>banco.bean.Extrato</class>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.dialect"
                value="org.hibernate.dialect.HSQLDialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

4.7 Administração da Infraestrutura de Chaves Públicas

icp.model.IIcpAdmin

```
package icp.model;

import java.io.InputStream;
import java.util.Date;
import java.util.List;
import javax.ejb.Local;

import icp.bean.Ca;
import icp.bean.Certificado;
import icp.bean.TipoCertificado;
import icp.bean.Usuario;

/**
 * Aplicacao para Gerenciamento de uma PKI (ICP) no padrao ICP-Brasil.
 * @author Robson Martins (robson@robsonmartins.com)
 * @see <a target="_blank"
 * href="http://www.iti.gov.br/twiki/bin/view/Certificacao/EstruturaIcp">
 * Estrutura da ICP-Brasil</a>
 */
@Local
public interface IIcpAdmin {
    /** Role para acessar a aplicacao como admininistrador. */
    public static final String ICPADMIN_ADMIN_ACCESS_ROLE = "admin";
    /**
     * Lista todas as AC Raiz existentes.
     * @return Lista com todas AC Raiz.
     */
    public List<Ca> listarACRaiz();

    /**
     * Cria uma nova AC Raiz.
     * @param nomeAC Nome da AC Raiz.
     * @param lcrURI Caminho (URI) da Lista de Certificados Revogados (LCR).
     * @param dpcURI Caminho (URI) do Documento com as Politicas
     *    de Certificacao (DPC).
     * @param acPassword Senha da AC Raiz, para criptografar a chave privada.
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @param subjCN Subject (DN): Common Name (CN).
     * @throws Exception
     */
```

icp.model.IIcpAdmin

```
public void adicionarACRaiz(String nomeAC, String lcrURI,
        String dpcURI, String acPassword, String subjC, String subjO,
        String subjOU, String subjCN) throws Exception;

/**
 * Remove uma AC Raiz.
 * @param nomeAC Nome da AC Raiz.
 * @throws Exception
 */
public void removerACRaiz(String nomeAC) throws Exception;

/**
 * Cria uma lista de certificados revogados (CRL), para uma AC Raiz.
 * @param acRaiz Objeto que representa a AC Raiz.
 * @param acPassword Senha da AC Raiz.
 * @throws Exception
 */
public void criarACRaizLCR(Ca acRaiz, String acPassword) throws Exception;

/**
 * Lista todas AC Intermediarias existentes.
 * @return Lista com todas AC Intermediarias.
 */
public List<Ca> listarACInterm();

/**
 * Cria uma nova AC Intermediaria.
 * @param acRaiz Objeto que representa a AC Raiz.
 * @param nomeAC Nome da AC.
 * @param lcrURI Caminho (URI) da Lista de Certificados Revogados (LCR).
 * @param dpcURI Caminho (URI) do Documento com as Politicas
 *   de Certificacao (DPC).
 * @param keyPassword Senha da AC, para criptografar a chave privada.
 * @param acRaizPassword Senha da AC Raiz.
 * @param subjC Subject (DN): Country (C).
 * @param subjO Subject (DN): Organization (O).
 * @param subjOU Subject (DN): Organizational Unit (OU).
 * @param subjCN Subject (DN): Common Name (CN).
 * @throws Exception
 */
public void adicionarACInterm(Ca acRaiz, String nomeAC, String lcrURI,
        String dpcURI, String keyPassword, String acRaizPassword,
        String subjC, String subjO, String subjOU, String subjCN)
                throws Exception;
```

icp.model.IIcpAdmin

```
    /**
     * Remove uma AC Intermediaria.
     * @param nomeAC Nome da AC Intermediaria.
     * @throws Exception
     */
    public void removerACInterm(String nomeAC) throws Exception;

    /**
     * Cria uma lista de certificados revogados (CRL), para uma AC Intermediaria.
     * @param ac Objeto que representa a AC Intermediaria.
     * @param acPassword Senha da AC Intermediaria.
     * @throws Exception
     */
    public void criarACIntermLCR(Ca ac, String acPassword) throws Exception;

    /**
     * Retorna uma lista com os nomes dos Tipos de Certificado disponiveis.
     * @return Lista com as descricoes dos Tipos de Certificado.
     */
    public List<TipoCertificado> listarTipoCert();

    /**
     * Retorna uma lista com todos os certificados emitidos por uma AC.
     * @param nomeAC Nome da AC emissora.
     * @return Lista com os certificados emitidos pela AC.
     */
    public List<Certificado> listarCert(String nomeAC);

    /**
     * Emite um certificado no padrao e-CPF.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param keyPassword Senha para criptografar a chave privada.
     * @param acPassword Senha da AC.
     * @param nome Nome do titular do e-CPF.
     * @param cpf Numero do CPF, sem pontuacao (11 numeros).
     * @param email Endereco de email.
     * @param nascimento Data de nascimento.
     * @param pisPasep Numero do PIS/PASEP/NIS, sem pontuacao (max. 11 numeros).
     * @param rg Numero do RG, alfanumerico, sem pontuacao (max. 15 caracteres).
     * @param rgOrgEmissor Orgao emissor do RG (max. 4 caracteres).
     * @param rgUF UF de emissao do RG (2 caracteres).
     * @param cei Numero do CEI (12 numeros).
     * @param titulo Numero do Titulo de Eleitor (max. 12 numeros).
     * @param tituloZona Zona Eleitoral do Titulo (max. 3 numeros).
     * @param tituloSecao Secao Eleitoral do Titulo (max. 4 numeros).
     * @param tituloMunicipio Municipio do Titulo Eleitoral (max. 20 caracteres).
     * @param tituloUF UF do Titulo Eleitoral (2 caracteres).
```

icp.model.IIcpAdmin

```
     * @param login Nome de Usuario (login de rede).
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @throws Exception
     */
    public void emitirCertEcpf(Ca acEmissora, String keyPassword,
            String acPassword,
            String nome, String cpf, String email, Date nascimento,
            String pisPasep, String rg, String rgOrgEmissor, String rgUF,
            String cei, String titulo, String tituloZona, String tituloSecao,
            String tituloMunicipio, String tituloUF, String login,
            String subjC, String subjO, String subjOU) throws Exception;

    /**
     * Emite um certificado no padrao e-CNPJ.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param keyPassword Senha para criptografar a chave privada.
     * @param acPassword Senha da AC.
     * @param nomePJ Nome empresarial.
     * @param nome Nome do responsavel.
     * @param cpf Numero do CPF do responsavel, sem pontuacao (11 numeros).
     * @param email Endereco de email do responsavel.
     * @param nascimento Data de nascimento do responsavel.
     * @param pisPasep Numero do PIS/PASEP/NIS do responsavel, sem pontuacao
     *   (max. 11 numeros).
     * @param rg Numero do RG do responsavel, alfanumerico, sem pontuacao
     *   (max. 15 caracteres).
     * @param rgOrgEmissor Orgao emissor do RG do responsavel (max. 4 caracteres).
     * @param rgUF UF de emissao do RG do responsavel (2 caracteres).
     * @param cei Numero do CEI do responsavel (12 numeros).
     * @param cnpj Numero do CNPJ, sem pontuacao (max. 14 numeros).
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @param subjL Subject (DN): Locality (L).
     * @param subjST Subject (DN): State (ST).
     * @throws Exception
     */
    public void emitirCertEcnpj(Ca acEmissora, String keyPassword,
            String acPassword,
            String nomePJ, String nome, String cpf,
            String email, Date nascimento, String pisPasep, String rg,
            String rgOrgEmissor, String rgUF, String cei,
            String cnpj, String subjC, String subjO, String subjOU,
            String subjL, String subjST) throws Exception;
```

icp.model.IIcpAdmin

```
    /**
     * Emite um certificado no padrao RIC.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param keyPassword Senha para criptografar a chave privada.
     * @param acPassword Senha da AC.
     * @param nome Nome do titular do e-CPF.
     * @param ric Numero do RIC, sem pontuacao (11 numeros).
     * @param cpf Numero do CPF, sem pontuacao (11 numeros).
     * @param email Endereco de email.
     * @param nascimento Data de nascimento.
     * @param pisPasep Numero do PIS/PASEP/NIS, sem pontuacao (max. 11 numeros).
     * @param rg Numero do RG, alfanumerico, sem pontuacao (max. 15 caracteres).
     * @param rgOrgEmissor Orgao emissor do RG (max. 4 caracteres).
     * @param rgUF UF de emissao do RG (2 caracteres).
     * @param cei Numero do CEI (12 numeros).
     * @param titulo Numero do Titulo de Eleitor (max. 12 numeros).
     * @param tituloZona Zona Eleitoral do Titulo (max. 3 numeros).
     * @param tituloSecao Secao Eleitoral do Titulo (max. 4 numeros).
     * @param tituloMunicipio Municipio do Titulo Eleitoral (max. 20 caracteres).
     * @param tituloUF UF do Titulo Eleitoral (2 caracteres).
     * @param login Nome de Usuario (login de rede).
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @throws Exception
     */
    public void emitirCertRic(Ca acEmissora, String keyPassword,
            String acPassword,
            String nome, String ric, String cpf, String email, Date nascimento,
            String pisPasep, String rg,    String rgOrgEmissor, String rgUF,
            String cei, String titulo, String tituloZona, String tituloSecao,
            String tituloMunicipio,    String tituloUF, String login,
            String subjC, String subjO, String subjOU) throws Exception;

    /**
     * Emite um certificado no padrao e-CODIGO.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param keyPassword Senha para criptografar a chave privada.
     * @param acPassword Senha da AC.
     * @param nomePJ Nome empresarial.
     * @param nome Nome do responsavel.
     * @param cpf Numero do CPF do responsavel, sem pontuacao (11 numeros).
     * @param email Endereco de email do responsavel.
     * @param nascimento Data de nascimento do responsavel.
     * @param pisPasep Numero do PIS/PASEP/NIS do responsavel, sem pontuacao
     *   (max. 11 numeros).
     * @param rg Numero do RG do responsavel, alfanumerico, sem pontuacao
```

icp.model.IIcpAdmin

```
     *   (max. 15 caracteres).
     * @param rgOrgEmissor Orgao emissor do RG do responsavel (max. 4 caracteres).
     * @param rgUF UF de emissao do RG do responsavel (2 caracteres).
     * @param cnpj Numero do CNPJ, sem pontuacao (max. 14 numeros).
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @throws Exception
     */
    public void emitirCertEcodigo(Ca acEmissora, String keyPassword,
            String acPassword,
            String nomePJ, String nome, String cpf,
            String email, Date nascimento, String pisPasep, String rg,
            String rgOrgEmissor, String rgUF,
            String cnpj, String subjC, String subjO, String subjOU)
                    throws Exception;

    /**
     * Emite um certificado no padrao e-SERVIDOR.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param keyPassword Senha para criptografar a chave privada.
     * @param acPassword Senha da AC.
     * @param nomeDNS Nome de DNS do servidor.
     * @param nomePJ Nome empresarial.
     * @param guid GUID do servidor.
     * @param nome Nome do responsavel.
     * @param cpf Numero do CPF do responsavel, sem pontuacao (11 numeros).
     * @param email Endereco de email do responsavel.
     * @param nascimento Data de nascimento do responsavel.
     * @param pisPasep Numero do PIS/PASEP/NIS do responsavel, sem pontuacao
     *   (max. 11 numeros).
     * @param rg Numero do RG do responsavel, alfanumerico, sem pontuacao
     *   (max. 15 caracteres).
     * @param rgOrgEmissor Orgao emissor do RG do responsavel (max. 4 caracteres).
     * @param rgUF UF de emissao do RG do responsavel (2 caracteres).
     * @param cnpj Numero do CNPJ, sem pontuacao (max. 14 numeros).
     * @param subjC Subject (DN): Country (C).
     * @param subjO Subject (DN): Organization (O).
     * @param subjOU Subject (DN): Organizational Unit (OU).
     * @throws Exception
     */
    public void emitirCertEservidor(Ca acEmissora, String keyPassword,
            String acPassword, String nomeDNS, String nomePJ, String guid,
            String nome, String cpf, String email, Date nascimento,
            String pisPasep, String rg,    String rgOrgEmissor, String rgUF,
            String cnpj, String subjC, String subjO, String subjOU)
                        throws Exception;
```

icp.model.IIcpAdmin

```
/**
 * Emite um certificado no padrao e-APLICACAO.
 * @param acEmissora Objeto que representa a AC emissora (issuer).
 * @param keyPassword Senha para criptografar a chave privada.
 * @param acPassword Senha da AC.
 * @param nomeApp Nome da aplicacao.
 * @param nomePJ Nome empresarial.
 * @param nome Nome do responsavel.
 * @param cpf Numero do CPF do responsavel, sem pontuacao (11 numeros).
 * @param email Endereco de email do responsavel.
 * @param nascimento Data de nascimento do responsavel.
 * @param pisPasep Numero do PIS/PASEP/NIS do responsavel, sem pontuacao
 *   (max. 11 numeros).
 * @param rg Numero do RG do responsavel, alfanumerico, sem pontuacao
 *   (max. 15 caracteres).
 * @param rgOrgEmissor Orgao emissor do RG do responsavel (max. 4 caracteres).
 * @param rgUF UF de emissao do RG do responsavel (2 caracteres).
 * @param cnpj Numero do CNPJ, sem pontuacao (max. 14 numeros).
 * @param subjC Subject (DN): Country (C).
 * @param subjO Subject (DN): Organization (O).
 * @param subjOU Subject (DN): Organizational Unit (OU).
 * @throws Exception
 */
public void emitirCertEaplicacao(Ca acEmissora, String keyPassword,
        String acPassword,
        String nomeApp, String nomePJ, String nome, String cpf,
        String email, Date nascimento, String pisPasep, String rg,
        String rgOrgEmissor, String rgUF,
        String cnpj, String subjC, String subjO, String subjOU)
                throws Exception;

/**
 * Revoga um certificado.
 * @param acEmissora Objeto que representa a AC emissora (issuer).
 * @param cert Certificado a ser revogado.
 * @param acPassword Senha da AC.
 * @throws Exception
 */
public void revogarCert(Ca acEmissora, Certificado cert,
        String acPassword) throws Exception;
```

icp.model.IIcpAdmin

```
    /**
     * Renova o certificado de uma AC Raiz.
     * @param acRaiz Objeto que representa a AC Raiz.
     * @param password Senha da AC Raiz.
     * @throws Exception
     */
    public void renovarACRaizCert(Ca acRaiz, String password) throws Exception;

    /**
     * Renova o certificado de uma AC Intermediaria.
     * @param ac Objeto que representa a AC Intermediaria.
     * @param password Senha da AC Intermediaria.
     * @throws Exception
     */
    public void renovarACIntermCert(Ca ac, String password) throws Exception;

    /**
     * Renova um certificado (reemissao).
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param cert Certificado a ser renovado.
     * @param acPassword Senha da AC.
     * @throws Exception
     */
    public void renovarCert(Ca acEmissora, Certificado cert,
            String acPassword) throws Exception;

    /**
     * Atualiza o status de todos os certificados expirados,
     *   no banco de dados.
     * @throws Exception
     */
    public void atualizarStatusCertExpirados();

    /**
     * Executa um backup dos arquivos de dados da ICP.
     * @param outFileName Nome do arquivo de backup a ser gerado.
     * @throws Exception
     */
    public void gerarBackup(String outFileName) throws Exception;

    /**
     * Retorna o diretorio base dos dados da ICP.
     * @return Caminho do diretorio base da ICP.
     */
    public String getBaseDir();
```

icp.model.IIcpAdmin

```
    /**
     * Retorna a lista de usuarios cadastrados.
     * @return Lista de usuarios.
     */
    public List<Usuario> listarUsuarios();

    /**
     * Retorna a lista de usuarios cadastrados, para a role especificada.
     * @param role Role dos usuarios.
     * @return Lista de usuarios.
     */
    public List<Usuario> listarUsuariosPorRole(String role);

    /**
     * Retorna um Usuario pelo DN.
     * @param dname Distinguished Name (DN) do Usuario no banco de dados.
     * @return Usuario ou null se nao encontrado.
     */
    public Usuario localizarUsuario(String dname);

    /**
     * Adiciona um usuario ao cadastro, a partir de um objeto
     *   {@link InputStream} (que pode apontar para um arquivo de certificado).
     * @param istream Objeto InputStream.
     * @param role Role do usuario a ser adicionado.
     * @return Usuario adicionado.
     * @throws Exception
     */
    public Usuario adicionarUsuario(InputStream istream, String role)
           throws Exception;

    /**
     * Adiciona um administrador ao cadastro, a partir de um objeto
     *   {@link InputStream} (que pode apontar para um arquivo de certificado).
     * @param istream Objeto InputStream.
     * @return Usuario adicionado.
     * @throws Exception
     */
    public Usuario adicionarAdmin(InputStream istream) throws Exception;

    /**
     * Remove um usuario do cadastro.
     * @param dname Distinguished Name (DN) do usuario no banco de dados.
     * @throws Exception
     */
    public void removerUsuario(String dname) throws Exception;
}
```

icp.model.IcpAdmin

```
package icp.model;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.MessageDigest;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Properties;
import javax.annotation.PostConstruct;
import javax.annotation.security.PermitAll;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.jboss.util.StringPropertyReplacer;
import sun.misc.BASE64Encoder;

import com.robsonmartins.fiap.tcc.util.CertificadoSerializador;
import sicid.bean.Cidadao;
import sicid.util.CertificadoIcpBrasilParser;
import sicid.ws.SICidClient;
import icp.bean.Ca;
import icp.bean.Certificado;
import icp.bean.StatusCertificado;
import icp.bean.TipoCertificado;
import icp.bean.Usuario;
import icp.dao.CaDAO;
import icp.dao.CertificadoDAO;
import icp.dao.IcpBrasilDAO;
import icp.dao.RootCaDAO;
import icp.dao.UsuarioDAO;
import icp.util.SystemWrapper;
```

icp.model.IcpAdmin

```
/**
 * Aplicacao para Gerenciamento de uma PKI (ICP) no padrao ICP-Brasil.
 * @author Robson Martins (robson@robsonmartins.com)
 * @see <a target="_blank"
 *    href="http://www.iti.gov.br/twiki/bin/view/Certificacao/EstruturaIcp">
 *    Estrutura da ICP-Brasil</a>
 */
@PermitAll
@Stateless
public class IcpAdmin implements IIcpAdmin {
    /* nome do arquivo de configuracao da aplicacao */
    private static final String ICP_PROPERTIES_FILE_NAME = "icpadmin.properties";
    /* nome do arquivo de properties com a configuracao de conexao ao sicid */
    private static final String ICPADMIN_SICID_CLIENT_PROPS_FILE =
        ICP_PROPERTIES_FILE_NAME;
    /* nome da unidade de persistencia configurada em persistence.xml */
    private static final String PERSISTENCE_UNIT_NAME = "icpadmin";
    /* caminho do diretorio base das AC's Raiz */
    private static final String ICP_ROOT_CA_BASE_DIR =
        File.separatorChar + "root";
    /* caminho do diretorio base das AC's Intermediarias */
    private static final String ICP_CA_BASE_DIR = File.separatorChar + "ca";
    /* Timeout para esperar termino de processo do Sistema Operacional, em ms */
    private static final long PROCESS_WAIT_TIMEOUT = 120000;
    /* diretorio base dos dados da ICP */
    private String baseDir;
    private RootCaDAO rootCaDAO;
    private CaDAO caDAO;
    private IcpBrasilDAO icpBrasilDAO;
    private CertificadoDAO certDAO;
    private UsuarioDAO usuarioDAO;
    /* Cliente SICid */
    private SICidClient sicidClient;
    /* EntityManager para JPA. */
    @PersistenceContext(unitName=PERSISTENCE_UNIT_NAME)
    private EntityManager entityManager;
    /* para fazer log */
    private static Logger logger;
    private static boolean trace;
```

icp.model.IcpAdmin

```
/**
 * Cria uma nova instancia do Gerenciador de ICP (PKI).
 * @throws Exception
 */
public IcpAdmin() {
    logger = LogManager.getLogger(IcpAdmin.class);
    trace = logger.isTraceEnabled();
}

/* inicializa DAO's */
@PostConstruct
protected void init() {
    Properties props = null;
    try {
        props = getPropertiesFromFile();
        baseDir = props.getProperty("icpFilesDir");
        /* substitui os valores das system props do JBoss */
        baseDir = StringPropertyReplacer.replaceProperties(baseDir);
    } catch (Exception e) {
        if (trace) {
            logger.error("Erro ao ler arquivo properties", e);
        }
        baseDir = "icpadmin";
    }
    rootCaDAO = new RootCaDAO(baseDir + ICP_ROOT_CA_BASE_DIR);
    caDAO = new CaDAO(baseDir + ICP_CA_BASE_DIR);
    icpBrasilDAO = new IcpBrasilDAO(rootCaDAO, caDAO);
    certDAO = new CertificadoDAO(entityManager);
    usuarioDAO = new UsuarioDAO(entityManager);
    sicidClient = new SICidClient();
}

@Override
public List<Ca> listarACRaiz() {
    return rootCaDAO.refreshListCA();
}

@Override
public void adicionarACRaiz(String nomeAC, String lcrURI,
        String dpcURI, String acPassword, String subjC, String subjO,
        String subjOU, String subjCN) throws Exception {

    icpBrasilDAO.createRootCA(nomeAC, lcrURI, dpcURI, acPassword, subjC,
                              subjO, subjOU, subjCN);
}
```

icp.model.IcpAdmin

```
@Override
public void removerACRaiz(String nomeAC) throws Exception {
    rootCaDAO.deleteCA(nomeAC);
    certDAO.excluirPorNomeAC(nomeAC);
}

@Override
public void criarACRaizLCR(Ca acRaiz, String acPassword) throws Exception {
    icpBrasilDAO.createCRLforRootCA(acRaiz, acPassword);
}

@Override
public List<Ca> listarACInterm() {
    return caDAO.refreshListCA();
}

@Override
public void adicionarACInterm(Ca acRaiz, String nomeAC, String lcrURI,
        String dpcURI, String keyPassword, String acRaizPassword,
        String subjC, String subjO, String subjOU, String subjCN)
                throws Exception {

    icpBrasilDAO.createCA(nomeAC, acRaiz, lcrURI, dpcURI, keyPassword,
            acRaizPassword, subjC, subjO, subjOU, subjCN);
}

@Override
public void removerACInterm(String nomeAC) throws Exception {
    caDAO.deleteCA(nomeAC);
    certDAO.excluirPorNomeAC(nomeAC);
}

@Override
public void criarACIntermLCR(Ca ac, String acPassword) throws Exception {
    icpBrasilDAO.createCRLforCA(ac, acPassword);
}

@Override
public List<TipoCertificado> listarTipoCert() {
    return Arrays.asList(TipoCertificado.values());
}

@Override
public List<Certificado> listarCert(String nomeAC) {
    return certDAO.listarPorNomeAC(nomeAC);
}
```

icp.model.IcpAdmin

```
@Override
public void emitirCertEcpf(Ca acEmissora, String keyPassword,
        String acPassword,
        String nome, String cpf, String email, Date nascimento,
        String pisPasep, String rg,    String rgOrgEmissor, String rgUF,
        String cei, String titulo, String tituloZona, String tituloSecao,
        String tituloMunicipio,    String tituloUF, String login,
        String subjC, String subjO, String subjOU) throws Exception {

    String certOutFileName = acEmissora.getUsrCertFile   ();
    String keyOutFileName  = acEmissora.getUsrKeyFile    ();
    String reqOutFileName  = acEmissora.getUsrReqFile    ();
    icpBrasilDAO.createEcpf(acEmissora, certOutFileName, keyOutFileName,
                            reqOutFileName, keyPassword, acPassword, nome,
                            cpf, email, nascimento, pisPasep, rg,
                            rgOrgEmissor, rgUF, cei, titulo, tituloZona,
                            tituloSecao, tituloMunicipio, tituloUF, login,
                            subjC, subjO, subjOU);
    String commonName =
        buildCnByTypeCert(TipoCertificado.ECPF, nome, null, null, null,
                          cpf, null);
    exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                       commonName);
    cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                  TipoCertificado.ECPF, commonName);
}

@Override
public void emitirCertEcnpj(Ca acEmissora, String keyPassword,
        String acPassword,
        String nomePJ, String nome, String cpf,
        String email, Date nascimento, String pisPasep, String rg,
        String rgOrgEmissor, String rgUF, String cei,
        String cnpj, String subjC, String subjO, String subjOU,
        String subjL, String subjST) throws Exception {

    String certOutFileName = acEmissora.getUsrCertFile();
    String keyOutFileName  = acEmissora.getUsrKeyFile ();
    String reqOutFileName  = acEmissora.getUsrReqFile ();
    icpBrasilDAO.createEcnpj(acEmissora, certOutFileName, keyOutFileName,
                             reqOutFileName, keyPassword, acPassword, nomePJ,
                             nome, cpf, email, nascimento, pisPasep, rg,
                             rgOrgEmissor, rgUF, cei, cnpj, subjC, subjOU,
                             subjOU, subjL, subjST);
    String commonName =
        buildCnByTypeCert(TipoCertificado.ECNPJ, nome, nomePJ, null, null,
                          cpf, cnpj);
```

icp.model.IcpAdmin

```
        exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                           commonName);
        cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                      TipoCertificado.ECNPJ, commonName);
    }

    @Override
    public void emitirCertRic(Ca acEmissora, String keyPassword,
            String acPassword,
            String nome, String ric, String cpf, String email, Date nascimento,
            String pisPasep, String rg, String rgOrgEmissor, String rgUF,
            String cei, String titulo, String tituloZona, String tituloSecao,
            String tituloMunicipio, String tituloUF, String login,
            String subjC, String subjO, String subjOU) throws Exception {

        String certOutFileName = acEmissora.getUsrCertFile();
        String keyOutFileName  = acEmissora.getUsrKeyFile ();
        String reqOutFileName  = acEmissora.getUsrReqFile ();
        icpBrasilDAO.createRic(acEmissora, certOutFileName, keyOutFileName,
                               reqOutFileName, keyPassword, acPassword, nome, ric,
                               cpf, email, nascimento, pisPasep, rg, rgOrgEmissor,
                               rgUF, cei, titulo, tituloZona, tituloSecao,
                               tituloMunicipio, tituloUF, login, subjC, subjO,
                               subjOU);
        String commonName =
            buildCnByTypeCert(TipoCertificado.RIC, nome, null, null, null,
                              cpf, null);
        exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                           commonName);
        cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                      TipoCertificado.RIC, commonName);
    }

    @Override
    public void emitirCertEcodigo(Ca acEmissora, String keyPassword,
            String acPassword,
            String nomePJ, String nome, String cpf,
            String email, Date nascimento, String pisPasep, String rg,
            String rgOrgEmissor, String rgUF,
            String cnpj, String subjC, String subjO, String subjOU)
                    throws Exception {

        String certOutFileName = acEmissora.getUsrCertFile();
        String keyOutFileName  = acEmissora.getUsrKeyFile ();
        String reqOutFileName  = acEmissora.getUsrReqFile ();
```

icp.model.IcpAdmin

```
        icpBrasilDAO.createEcodigo(acEmissora, certOutFileName, keyOutFileName,
                                   reqOutFileName, keyPassword, acPassword,
                                   nomePJ, nome, cpf, email, nascimento, pisPasep,
                                   rg, rgOrgEmissor, rgUF, cnpj, subjC, subjO,
                                   subjOU);
        String commonName =
            buildCnByTypeCert(TipoCertificado.ECODIGO, nome, nomePJ, null, null,
                              cpf, cnpj);
        exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                           commonName);
        cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                      TipoCertificado.ECODIGO, commonName);
    }

    @Override
    public void emitirCertEservidor(Ca acEmissora, String keyPassword,
            String acPassword,
            String nomeDNS, String nomePJ, String guid,
            String nome, String cpf, String email, Date nascimento,
            String pisPasep, String rg, String rgOrgEmissor, String rgUF,
            String cnpj, String subjC, String subjO, String subjOU)
                    throws Exception {

        String certOutFileName = acEmissora.getUsrCertFile();
        String keyOutFileName  = acEmissora.getUsrKeyFile ();
        String reqOutFileName  = acEmissora.getUsrReqFile ();
        icpBrasilDAO.createEservidor(acEmissora, certOutFileName, keyOutFileName,
                                     reqOutFileName, keyPassword, acPassword,
                                     nomeDNS, nomePJ, guid, nome, cpf, email,
                                     nascimento, pisPasep, rg, rgOrgEmissor, rgUF,
                                     cnpj, subjC, subjO, subjOU);
        String commonName =
            buildCnByTypeCert(TipoCertificado.ESERVIDOR, nome, nomePJ, nomeDNS,
                              null, cpf, cnpj);
        exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                           commonName);
        cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                      TipoCertificado.ESERVIDOR, commonName);
    }

    @Override
    public void emitirCertEaplicacao(Ca acEmissora, String keyPassword,
            String acPassword, String nomeApp, String nomePJ, String nome,
            String cpf, String email, Date nascimento, String pisPasep, String rg,
            String rgOrgEmissor, String rgUF, String cnpj, String subjC,
            String subjO, String subjOU) throws Exception {
```

icp.model.IcpAdmin

```
        String certOutFileName = acEmissora.getUsrCertFile();
        String keyOutFileName  = acEmissora.getUsrKeyFile ();
        String reqOutFileName  = acEmissora.getUsrReqFile ();
        icpBrasilDAO.createEaplicacao(acEmissora, certOutFileName, keyOutFileName,
                                      reqOutFileName, keyPassword, acPassword,
                                      nomeApp, nomePJ, nome, cpf, email,
                                      nascimento, pisPasep, rg, rgOrgEmissor,
                                      rgUF, cnpj, subjC, subjO, subjOU);
        String commonName =
            buildCnByTypeCert(TipoCertificado.EAPLICACAO, nome, nomePJ, null,
                              nomeApp, cpf, cnpj);
        exportarCertPkcs12(certOutFileName, keyOutFileName, keyPassword,
                           commonName);
        cadastrarCert(acEmissora, certOutFileName, keyOutFileName, reqOutFileName,
                      TipoCertificado.EAPLICACAO, commonName);
    }

    @Override
    public void revogarCert(Ca acEmissora, Certificado cert,
            String acPassword) throws Exception {

        caDAO.revokeCert(acEmissora,
                baseDir + File.separatorChar + cert.getCertFilename(),
                acPassword);
        revogarCadastroCert(cert);
        if (acEmissora.getRoot() == null || "".equals(acEmissora.getRoot())) {
            icpBrasilDAO.createCRLforRootCA(acEmissora, acPassword);
        } else {
            icpBrasilDAO.createCRLforCA(acEmissora, acPassword);
        }
    }

    @Override
    public void renovarACRaizCert(Ca acRaiz, String password) throws Exception {
        icpBrasilDAO.renewRootCaCert(acRaiz, password);
    }

    @Override
    public void renovarACIntermCert(Ca ac, String password) throws Exception {
        icpBrasilDAO.renewCaCert(ac, password);
    }
```

icp.model.IcpAdmin

```
    @Override
    public void renovarCert(Ca acEmissora, Certificado cert,
            String acPassword) throws Exception {

        icpBrasilDAO.renewCert(acEmissora,
                baseDir + File.separatorChar + cert.getCertFilename(),
                baseDir + File.separatorChar + cert.getReqFilename(),
                acPassword);
        renovarCadastroCert(cert);
    }

    @Override
    public void atualizarStatusCertExpirados() {
        certDAO.atualizarStatusCertExpirados();
    }

    @Override
    public void gerarBackup(String outFileName) throws Exception {
        String icpDirName = new File(baseDir).getName();
        String[] command = { "tar", "-czf", outFileName, "-C", baseDir,
                ".." + File.separatorChar + icpDirName };
        Process p = SystemWrapper.executeCommand(command);
        if (SystemWrapper.processWait(p, PROCESS_WAIT_TIMEOUT) != 0) {
            throw new Exception(String.format(
                    "Error in tar command: %s",
                    SystemWrapper.getProcessErrorStr(p)));
        }
    }

    @Override
    public String getBaseDir() {
        return baseDir;
    }

    @Override
    public List<Usuario> listarUsuarios() {
        return usuarioDAO.listar();
    }

    @Override
    public List<Usuario> listarUsuariosPorRole(String role) {
        return usuarioDAO.listarPorRole(role);
    }
```

icp.model.IcpAdmin

```
@Override
public Usuario localizarUsuario(String dname) {
    return usuarioDAO.localizar(dname);
}

@Override
public Usuario adicionarUsuario(InputStream istream,
        String role) throws Exception {

    Usuario usuario = null;
    try {
        if (trace) {
            logger.trace("Adicionando usuario");
        }
        X509Certificate x509cert =
            CertificadoSerializador.loadCertFromStream(istream);
        String dname = x509cert.getSubjectX500Principal().getName();
        usuario = new Usuario();
        usuario.setDname(dname);
        MessageDigest md = MessageDigest.getInstance("SHA1");
        String id =
            new BASE64Encoder().encode(
                    md.digest(dname.getBytes()));
        usuario.setUsername(id);
        String name = null;
        try {
            String content = CertificadoSerializador.certToStr(x509cert);
            sicidClient.connect(ICPADMIN_SICID_CLIENT_PROPS_FILE);
            Cidadao cidadao =
                sicidClient.getService().consultarCidadao(content);
            if (cidadao != null) {
                name = cidadao.getNome();
            }
            if (name == null) {
                throw new Exception("Cidad\u00E3o n\u00E3o encontrado.");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao obter o nome do cidadao", e);
            }
        }
        if (name == null) {
            name = CertificadoIcpBrasilParser.extractNomeFromCN(dname);
            if (name == null) { name = dname; }
        }
        usuario.setName(name);
        usuario.setRole(role);
```

icp.model.IcpAdmin

```
            usuarioDAO.inserir(usuario);
            if (trace) {
                logger.trace("Usuario adicionado com sucesso");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao adicionar usuario", e);
            }
            throw e;
        }
        return usuario;
    }

    @Override
    public Usuario adicionarAdmin(InputStream istream) throws Exception {
        return adicionarUsuario(istream, ICPADMIN_ADMIN_ACCESS_ROLE);
    }

    @Override
    public void removerUsuario(String dname) throws Exception {
        try {
            if (trace) {
                logger.trace("Removendo usuario");
            }
            usuarioDAO.excluir(dname);
            if (trace) {
                logger.trace("Usuario removido com sucesso");
            }
        } catch (Exception e) {
            if (trace) {
                logger.error("Erro ao remover usuario", e);
            }
            throw e;
        }
    }

    /* Exporta um certificado como PKCS#12 e sua chave privada num
     *  arquivo descriptografado.
     * @param certInFileName Nome do arquivo de certificado PEM.
     * @param keyInFileName Nome do arquivo de chave privada (key) criptografada.
     * @param keyPassword Senha da chave privada.
     * @param keyAlias Nome amigavel do par de chaves dentro do arquivo PKCS#12.
     * @throws Exception
     */
    private void exportarCertPkcs12(String certInFileName, String keyInFileName,
            String keyPassword, String keyAlias) throws Exception {
```

icp.model.IcpAdmin

```
        String pfxOutFileName    = certInFileName.replace(".pem.cer",".pfx");
        String keyNoPassFileName = keyInFileName .replace(".pem", ".plain");
        caDAO.exportCertPkcs12(certInFileName, keyInFileName, pfxOutFileName,
                               keyPassword, keyAlias);
        caDAO.exportKeyNoPassword(keyInFileName, keyNoPassFileName, keyPassword);
    }

    /* Insere um novo certificado no cadastro.
     * @param acEmissora Objeto que representa a AC emissora (issuer).
     * @param certFileName Nome do arquivo de certificado a ser gerado.
     * @param keyFileName Nome do arquivo de chave privada a ser gerado.
     * @param reqFileName Nome do arquivo de requisicao a ser gerado.
     * @param tipo Tipo do certificado.
     * @param commonName Common Name (CN) do certificado.
     * @throws Exception
     */
    private void cadastrarCert(Ca acEmissora, String certFileName,
            String keyFileName, String reqFileName, TipoCertificado tipo,
            String commonName) throws Exception {

        Certificado cert = new Certificado();
        cert.setId           (0);
        cert.setCertFilename(certFileName.replace(baseDir+File.separatorChar,""));
        cert.setKeyFilename (keyFileName .replace(baseDir+File.separatorChar,""));
        cert.setReqFilename (reqFileName .replace(baseDir+File.separatorChar,""));
        cert.setNomeAC       (acEmissora.getName());
        cert.setCommonName   (commonName);
        cert.setTipo         (tipo);
        cert.setStatus       (StatusCertificado.VALIDO);
        cert.setEmissao      (new Date());
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(cert.getEmissao());
        calendar.add(Calendar.DAY_OF_MONTH, icpBrasilDAO.getUserExpireDays());
        cert.setExpiracao(calendar.getTime());
        certDAO.inserir(cert);
    }

    /* Constroi o Common Name (CN), a partir do tipo de certificado.
     * @param tipo Tipo do certificado.
     * @param nome Nome do responsavel.
     * @param nomePJ Nome empresarial.
     * @param nomeDNS Nome de DNS do servidor.
     * @param nomeApp Nome da aplicacao.
     * @param cpf Numero do CPF do responsavel, sem pontuacao (11 numeros).
     * @param cnpj Numero do CNPJ, sem pontuacao (max. 14 numeros).
     * @return Common Name do certificado.
     */
```

icp.model.IcpAdmin

```
    private String buildCnByTypeCert(TipoCertificado tipo,
            String nome, String nomePJ, String nomeDNS, String nomeApp,
            String cpf, String cnpj) {

        StringBuilder cn = new StringBuilder();
        switch (tipo) {
            case ECPF      : cn.append(nome   ).append(':').append(cpf ); break;
            case ECNPJ     : cn.append(nomePJ ).append(':').append(cnpj); break;
            case RIC       : cn.append(nome   ).append(':').append(cpf ); break;
            case ECODIGO   : cn.append(nomePJ ).append(':').append(cnpj); break;
            case ESERVIDOR : cn.append(nomeDNS)                         ; break;
            case EAPLICACAO: cn.append(nomeApp).append(':').append(cnpj); break;
            default        :                                              break;
        }
        return cn.toString();
    }

    /* Atualiza um certificado no cadastro, no caso, uma revogacao.
     * @param cert Certificado sendo revogado.
     * @throws Exception
     */
    private void revogarCadastroCert(Certificado cert) throws Exception {
        cert.setStatus(StatusCertificado.REVOGADO);
        certDAO.inserir(cert);
    }

    /* Atualiza um certificado no cadastro, no caso, uma renovacao.
     * @param cert Certificado sendo renovado.
     * @throws Exception
     */
    private void renovarCadastroCert(Certificado cert) throws Exception {
        cert.setEmissao(new Date());
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(cert.getEmissao());
        calendar.add(Calendar.DAY_OF_MONTH, icpBrasilDAO.getUserExpireDays());
        cert.setExpiracao(calendar.getTime());
        cert.setStatus(StatusCertificado.VALIDO);
        certDAO.inserir(cert);
    }

    /* Obtem arquivo de properties com a configuracao da aplicacao.
     * @return Objeto contendo as properties do arquivo.
     * @throws Exception
     */
    private Properties getPropertiesFromFile() throws Exception {
        Properties props = new Properties();
        InputStream inputStream = null;
```

icp.model.IcpAdmin

```
        String propDir = null;
        /* tenta obter path "conf" do JBoss */
        propDir = System.getProperty("jboss.server.config.url");
        if (propDir == null) {
            /* tenta obter path "serverhome"/conf do JBoss */
            propDir = System.getProperty("jboss.server.home.url");
            if (propDir != null) { propDir += File.separatorChar + "conf"; }
        }
        if (propDir != null) {
            /* tenta obter arquivo no path especificado */
            try {
                inputStream =
                    this.getClass().getClassLoader().getResourceAsStream(
                            String.format("%s%s%s", propDir,
                                File.separator, ICP_PROPERTIES_FILE_NAME));
            } catch (Exception e) { }
        }
        if (inputStream == null) {
            /* se nao achou no dir JBoss, tenta obter arquivo via classloader */
            try {
                inputStream =
                    this.getClass().getClassLoader().getResourceAsStream(
                            ICP_PROPERTIES_FILE_NAME);
            } catch (Exception e) {    }
        }
        if (inputStream == null) {
            /* tenta obter arquivo no path META-INF especificado */
            propDir = "META-INF";
            try {
                inputStream =
                    this.getClass().getClassLoader().getResourceAsStream(
                            String.format("%s%s%s", propDir,
                                File.separator, ICP_PROPERTIES_FILE_NAME));
            } catch (Exception e) { }
        }
        if (inputStream == null) {
            throw new FileNotFoundException(
                    String.format("File %s not found.",
                                ICP_PROPERTIES_FILE_NAME));
        }
        props.load(inputStream);
        return props;
    }
}
```

icp.util.OpenSSLWrapper

```
package icp.util;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

/**
 * Encapsula funcionalidades proporcionadas pelo OpenSSL.
 * @author Robson Martins (robson@robsonmartins.com)
 * @see <a href="http://www.openssl.org/">OpenSSL Oficial Site</a>
 */
public class OpenSSLWrapper {
    /* Comando executavel do OpenSSL */
    private static final String[] OPENSSL_COMMAND_UNIX = {"openssl"};
    private static final String[] OPENSSL_COMMAND_WIN  = {"cmd", "/c", "openssl"};
    /* objeto para realizar log das operacoes */
    private static Logger logger;
    private static boolean trace;

    /* Inicializa atributos estaticos. */
    static {
        logger = LogManager.getLogger(OpenSSLWrapper.class);
        trace = logger.isTraceEnabled();
    }

    /**
     * Anexa um campo (field) ao subject (DN - Distinguished Name).
     * @param subj Subject a ter o campo anexado. Se null, sera' inicializado um
     *    subject vazio.
     * @param field Nome do campo a ser anexado ao subject (ex: "C", "O", "OU",
     *    etc.).
     * @param value Valor do campo a ser anexado ao subject.
     *   Pode ser uma string contendo valores separados por ponto-e-virgula, para
     *   anexar multiplos valores a um mesmo campo.
     * @return Objeto StringBuilder que representa o subject, com o campo anexado.
     * @see <a href="http://www.x500standard.com/">X500 Standard</a>
     */
    public static StringBuilder subjectAppend(StringBuilder subj, String field,
            String value) {
        if (subj == null) subj = new StringBuilder();
        if (field != null && value != null) {
            String[] values = value.split(";");
            for (String v : values) {
                subj.append(String.format("/%s=%s", field, v));
            }
```

icp.util.OpenSSLWrapper

```
        }
        return subj;
    }

    /**
     * Retorna os valores de um campo (field) de um subject
     *    (DN - Distinguished Name).
     * @param subject String representando um subject.
     * @param field Nome do campo a ser retornado (ex: "C", "O", "OU", etc.).
     * @return Valores atribuidos ao campo, ou vazio se nenhum.
     * @see <a href="http://www.x500standard.com/">X500 Standard</a>
     */
    public static List<String> subjectParse(String subject, String field) {
        List<String> subjList = new ArrayList<String>();
        if (subject != null && !"".equals(subject)) {
            String pattern = String.format("/%s=", field);
            int beginIndex = 0;
            int endIndex = 0;
            do {
                beginIndex = subject.indexOf(pattern, beginIndex);
                if (beginIndex >= 0) {
                    beginIndex += pattern.length();
                    endIndex = subject.indexOf("/", beginIndex);
                    if (endIndex > - 0)
                        subjList.add(subject.substring(beginIndex, endIndex));
                    else
                        subjList.add(subject.substring(beginIndex));
                }
            } while (beginIndex >= 0);
        }
        return subjList;
    }

    /**
     * Cria um certificado auto-assinado.
     * @param configFile Nome do arquivo de configuracao a ser usado.
     * @param keyOutFile Nome do arquivo de chave (key) a ser gerado.
     * @param certOutFile Nome do arquivo de certificado (PEM) a ser gerado.
     * @param expireDays Quantidade de dias para expirar o certificado.
     * @param keySize Tamanho da chave (RSA), em bits (ex: 1024, 2048, 4096).
     * @param algHash Algoritmo de Hash (ex: "sha1", "sha256").
     * @param password Senha para o certificado.
     * @param subject Subject (DN) para o certificado.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
```

icp.util.OpenSSLWrapper

```
public static Process createAutoSignCert(String configFile, String keyOutFile,
        String certOutFile, int expireDays, int keySize, String algHash,
        String password, String subject) throws IOException {

    String[] args =
        { "req", "-new", "-x509", "-config",  configFile ,
          "-keyout",  keyOutFile , "-out",  certOutFile ,
          "-days", String.valueOf(expireDays),
          "-newkey", "rsa:" + String.valueOf(keySize), "-" + algHash,
          "-passin", "pass:" + password ,
          "-passout", "pass:" + password ,
          "-subj",  subject  };
    return executeOpenSSL(args);
}

/**
 * Cria uma requisicao (PKCS#10).
 * @param configFile Nome do arquivo de configuracao a ser usado.
 * @param keyOutFile Nome do arquivo de chave (key) a ser gerado.
 * @param reqOutFile Nome do arquivo de requisicao (PEM) a ser gerado.
 * @param keySize Tamanho da chave (RSA), em bits (ex: 1024, 2048, 4096).
 * @param algHash Algoritmo de Hash (ex: "sha1", "sha256").
 * @param password Senha para a requisicao de certificado.
 * @param subject Subject (DN) para o certificado.
 * @return Objeto Process, representando o processo do OpenSSL chamado.
 * @throws IOException
 */
public static Process createCertReq(String configFile, String keyOutFile,
        String reqOutFile, int keySize, String algHash,
        String password, String subject) throws IOException {

    String[] args =
        { "req", "-outform", "PEM", "-out",  reqOutFile , "-verify", "-new",
          "-newkey", "rsa:" + String.valueOf(keySize), "-keyform", "PEM",
          "-keyout",  keyOutFile , "-" + algHash,
          "-passin", "pass:" + password ,
          "-passout", "pass:" + password ,
          "-config",  configFile ,
          "-subj",  subject  };
    return executeOpenSSL(args);
}
```

icp.util.OpenSSLWrapper

```
    /**
     * Importa uma requisicao de certificado (PKCS#10).
     * @param configFile Nome do arquivo de configuracao a ser usado.
     * @param reqInFile Nome do arquivo de requisicao a ser importado.
     * @param reqFormat Formato do arquivo de requisicao a ser importado
     *   ("PEM" ou "DER").
     * @param reqOutFile Nome do arquivo de requisicao (PEM) a ser gerado.
     * @param password Senha da requisicao de certificado.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process importCertReq(String configFile, String reqInFile,
            String reqFormat, String reqOutFile, String password)
                            throws IOException {
        String[] args =
            { "req", "-inform", reqFormat, "-outform", "PEM",
              "-in",  reqInFile ,
              "-passin", "pass:" + password ,
              "-passout", "pass:" + password ,
              "-out",  reqOutFile , "-verify",
              "-config",  configFile  };
        return executeOpenSSL(args);
    }

    /**
     * Cria (assina) um certificado (PEM) a partir de uma requisicao (PKCS#10).
     * @param configFile Nome do arquivo de configuracao a ser usado.
     * @param reqInFile Nome do arquivo de requisicao.
     * @param caKeyFile Nome do arquivo de chave (key) da CA.
     * @param caCertFile Nome do arquivo de certificado da CA
     *   (de quem emite [issuer] este certificado).
     * @param certOutFile Nome do arquivo de certificado (PEM) a ser gerado.
     * @param expireDays Quantidade de dias para expirar o certificado.
     * @param algHash Algoritmo de Hash (ex: "sha1", "sha256").
     * @param caPassword Senha do certificado da CA (issuer).
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process createCertFromReq(String configFile, String reqInFile,
            String caKeyFile, String caCertFile, String certOutFile,
            int expireDays, String algHash, String caPassword)
                            throws IOException {
        String[] args =
            { "ca", "-config",  configFile , "-md", algHash,
              "-out",  certOutFile ,
              "-key",  caPassword , "-in",  reqInFile ,
              "-days", String.valueOf(expireDays),
```

icp.util.OpenSSLWrapper

```
            "-keyfile",  caKeyFile , "-cert",  caCertFile , "-batch" };
        return executeOpenSSL(args);
    }

    /**
     * Revoga um certificado (PEM) ja' emitido.
     * @param configFile Nome do arquivo de configuracao a ser usado.
     * @param certInFile Nome do arquivo de certificado a ser revogado.
     * @param caCertFile Nome do arquivo de certificado da CA
     *   (de quem emitiu [issuer] este certificado).
     * @param caKeyFile Nome do arquivo de chave (key) da CA.
     * @param caPassword Senha do certificado da CA (issuer).
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process revokeCert(String configFile, String certInFile,
            String caCertFile, String caKeyFile, String caPassword)
                                throws IOException {
        String[] args =
            { "ca", "-revoke",  certInFile , "-config",  configFile ,
              "-cert",  caCertFile , "-keyfile",  caKeyFile ,
              "-key",  caPassword  };
        return executeOpenSSL(args);
    }

    /**
     * Converte um certificado PEM para DER.
     * @param pemCertFile Nome do arquivo do certificado (PEM) de origem.
     * @param derCertFile Nome do arquivo do certificado (DER) de destino.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process convertCertPem2Der(String pemCertFile,
            String derCertFile) throws IOException {

        String[] args =
            { "x509", "-inform", "PEM", "-outform", "DER", "-in",  pemCertFile ,
              "-out",  derCertFile };
        return executeOpenSSL(args);
    }
```

icp.util.OpenSSLWrapper

```
    /**
     * Exporta um certificado para o formato PKCS#12 (.pfx).
     * @param certInFile Nome do arquivo de certificado a ser convertido.
     * @param keyInFile Nome do arquivo de chave (key).
     * @param pfxCertFile Nome do arquivo de certificado (PKCS#12) de destino.
     * @param password Senha do certificado.
     * @param alias Nome amigavel do par de chaves dentro do arquivo PKCS#12.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process exportCertPkcs12(String certInFile, String keyInFile,
            String pfxCertFile, String password, String alias)
                                throws IOException {
        String[] args =
            { "pkcs12", "-export", "-inkey",  keyInFile,
              "-passin", "pass:" + password,
              "-passout", "pass:" + password,
              "-in",  certInFile, "-out",  pfxCertFile,
              "-name", alias };
        return executeOpenSSL(args);
    }

    /**
     * Exporta uma chave privada para um arquivo descriptografado (sem senha).
     * @param keyInFile Nome do arquivo de chave privada (key) criptografada.
     * @param keyOutFile Nome do arquivo de chave (key) a ser gerado.
     * @param password Senha da chave privada criptografada.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process exportKeyNoPassword(String keyInFile, String keyOutFile,
            String password) throws IOException {

        String[] args =
            { "rsa", "-in", keyInFile, "-out", keyOutFile,
              "-passin", "pass:" + password };
        return executeOpenSSL(args);
    }

    /**
     * Exporta uma cadeia de certificados como arquivo P7B.
     * @param certInFiles Nome dos arquivos de certificado (PEM) que
     *   formam a cadeia.
     * @param p7bCertFile Nome do arquivo de cadeia (P7B) a ser gerado.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
```

icp.util.OpenSSLWrapper

```
public static Process exportCertChain(String[] certInFiles,
        String p7bCertFile) throws IOException {

    String[] args = new String[4 + 2 * certInFiles.length];
    int argsIdx = 0;
    args[argsIdx++] = "crl2pkcs7";
    args[argsIdx++] = "-nocrl";
    for (int i = 0; i < certInFiles.length; i++) {
        args[argsIdx++] = "-certfile";
        args[argsIdx++] = certInFiles[i];
    }
    args[argsIdx++] = "-out";
    args[argsIdx++] = p7bCertFile;
    return executeOpenSSL(args);
}

/**
 * Gera uma lista de certificados revogados (CRL).
 * @param configFile Nome do arquivo de configuracao a ser usado.
 * @param caCertFile Nome do arquivo de certificado.
 * @param crlOutFile Nome do arquivo CRL a ser gerado.
 * @param crlDays Quantidade de dias para expirar a lista.
 * @param algHash Algoritmo de Hash (ex: "sha1", "sha256").
 * @param password Senha do certificado.
 * @return Objeto Process, representando o processo do OpenSSL chamado.
 * @throws IOException
 */
public static Process generateCRL(String configFile, String caCertFile,
        String crlOutFile, int crlDays, String algHash, String password)
                        throws IOException {
    String[] args =
        { "ca", "-config",  configFile , "-gencrl",
          "-crldays", String.valueOf(crlDays), "-md", algHash,
          "-out",  crlOutFile ,
          "-cert",  caCertFile , "-key",  password  };
    return executeOpenSSL(args);
}

/**
 * Retorna informacoes sobre uma lista de certificados revogados (CRL):
 *   campo lastUpdate.
 * @param crlInFile Nome do arquivo CRL.
 * @return Objeto Process, representando o processo do OpenSSL chamado.
 * @throws IOException
 */
public static Process getCRLInfoLastUpdate(String crlInFile)
                        throws IOException {
```

icp.util.OpenSSLWrapper

```
        String[] args =
            { "crl", "-inform", "PEM", "-noout",
              "-lastupdate", "-in",  crlInFile  };
        return executeOpenSSL(args);
    }

    /**
     * Retorna informacoes sobre uma lista de certificados revogados (CRL):
     *   campo nextUpdate.
     * @param crlInFile Nome do arquivo CRL.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    public static Process getCRLInfoNextUpdate(String crlInFile)
                              throws IOException {
        String[] args =
            { "crl", "-inform", "PEM", "-noout",
              "-nextupdate", "-in",  crlInFile  };
        return executeOpenSSL(args);
    }

    /* Executa o OpenSSL e retorna o processo de sistema associado.
     * @param args Argumentos de linha de comando para o OpenSSL.
     * @return Objeto Process, representando o processo do OpenSSL chamado.
     * @throws IOException
     */
    private static Process executeOpenSSL(String[] args) throws IOException {
        String[] openssl;
        String os = System.getProperty("os.name");
        if (os != null && os.toLowerCase().startsWith("windows")) {
            openssl = OPENSSL_COMMAND_WIN;
        } else {
            openssl = OPENSSL_COMMAND_UNIX;
        }
        String[] command = new String[openssl.length + args.length];
        System.arraycopy(openssl, 0, command, 0, openssl.length);
        System.arraycopy(args,  0, command, openssl.length, args.length);
        StringBuilder commandLine = new StringBuilder();
        for (String s : command) {
            if (commandLine.length() > 0) { commandLine.append(' '); }
            commandLine.append(s);
        }
```

icp.util.OpenSSLWrapper

```
        if (trace) {
            logger.trace(String.format("OpenSSL command-line: %s",
                                       commandLine.toString()));
        }
        return SystemWrapper.executeCommand(command);
    }
}
```

icp.util.SystemWrapper

```
package icp.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Collection;
import java.util.TreeSet;
import java.util.concurrent.TimeoutException;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;

/**
 * Encapsula funcionalidades do Sistema Operacional.
 * @author Robson Martins (robson@robsonmartins.com)
 */
public class SystemWrapper {
    /* Objeto Runtime para execucao de comandos no S.O. */
    private static final Runtime runtime = Runtime.getRuntime();
    /* Comando executavel para remover diretorios e arquivos recursivamente */
    private static final String[] RM_COMMAND_UNIX =
        { "rm", "-Rf" };
    private static final String[] RM_COMMAND_WIN  =
        { "cmd", "/c", "rd", "/s", "/q" };
    /* objeto para realizar log das operacoes */
    private static Logger logger;
    private static boolean trace;

    /* Inicializa atributos estaticos. */
    static {
        logger = LogManager.getLogger(SystemWrapper.class);
        trace = logger.isTraceEnabled();
    }

    /**
     * Executa um comando no sistema operacional.
     * @param command Array de Strings contendo o comando a ser executado
     *    e seus argumentos.
     * @return Objeto Process, representando o processo do comando chamado.
     * @throws IOException
     */
    public static Process executeCommand(String[] command) throws IOException {
        if (trace) {
            StringBuilder commandLine = new StringBuilder();
            for (String s : command) {
                if (commandLine.length() > 0) { commandLine.append(' '); }
```

icp.util.SystemWrapper

```
                commandLine.append(s);
            }
            logger.trace(String.format("Execute: %s", commandLine.toString()));
        }
        return runtime.exec(command);
    }

    /**
     * Espera pelo termino de um processo e retorna seu codigo de saida.
     * @param p Objeto Process, que representa um processo.
     * @return Codigo de saida do processo.
     * @throws Exception
     */
    public static int processWait(Process p, long timeout) throws Exception {
        int exitValue = 0;
        boolean exitOK = false;
        long start = System.currentTimeMillis();
        while (!exitOK) {
            try {
                exitValue = p.exitValue();
                exitOK = true;
            } catch (IllegalThreadStateException e) {
                Thread.sleep(10);
            }
            if (!exitOK && System.currentTimeMillis() >= (start + timeout)) {
                throw new TimeoutException("Timeout: process not terminated.");
            }
        }
        return exitValue;
    }

    /**
     * Retorna uma lista com os nomes dos subdiretorios contidos em um diretorio.
     * @param dir Caminho do diretorio a ser listado.
     * @return Lista com os nomes dos subdiretorios contidos em <i>dir</i>.
     */
    public static Collection<String> lsDir(String dir) {
        Collection<String> resultList = new TreeSet<String>();
        File rootDir = new File(dir);
        if (rootDir != null && rootDir.isDirectory()) {
            File[] subDirs = rootDir.listFiles(new FileFilter() {
                @Override
                public boolean accept(File pathname) {
                    return (pathname.isDirectory()
                            && !".".equals(pathname.getName())
                            && !"..".equals(pathname.getName()));
                }
```

icp.util.SystemWrapper

```
            });
            for (File subDir : subDirs) {
                resultList.add(subDir.getName());
            }
        }
        return resultList;
    }

    /**
     * Cria um diretorio no sistema de arquivos (nao recursivo).
     * @param dir Caminho do diretorio a ser criado
     * @return True se diretorio foi criado, false se houve erro.
     */
    public static boolean mkDir(String dir) {
        File rootDir = new File(dir);
        return rootDir.mkdir();
    }

    /**
     * Remove um diretorio e todo seu conteudo (recursivo).
     * @param dir Caminho do diretorio a ser removido.
     * @return Objeto Process, representando o processo do comando chamado.
     * @throws IOException
     */
    public static Process rmDir(String dir) throws IOException {
        String[] command;
        String os = System.getProperty("os.name");
        if (os != null && os.toLowerCase().startsWith("windows")) {
            command = new String[RM_COMMAND_WIN.length + 1];
            System.arraycopy(
                RM_COMMAND_WIN, 0, command, 0, RM_COMMAND_WIN.length);
            dir = dir.replace('/', '\\');
        } else {
            command = new String[RM_COMMAND_UNIX.length + 1];
            System.arraycopy(
                RM_COMMAND_UNIX, 0, command, 0, RM_COMMAND_UNIX.length);
        }
        command[command.length - 1] = dir;
        if (trace) {
            StringBuilder commandLine = new StringBuilder();
            for (String s : command) {
                if (commandLine.length() > 0) { commandLine.append(' '); }
                commandLine.append(s);
            }
            logger.trace(String.format("RMDIR command-line: %s",
                                       commandLine.toString()));
        }
```

icp.util.SystemWrapper

```
        return executeCommand(command);
    }

    /**
     * Retorna uma String contendo a saida do console de erro de
     *    um processo (stderr).
     * @param p Objeto Process, que representa um processo.
     * @return Saida do console de erro (stderr).
     * @throws IOException
     */
    public static String getProcessErrorStr(Process p) throws IOException {
        BufferedReader in =
            new BufferedReader(new InputStreamReader(p.getErrorStream()));
        StringBuilder errorStr = new StringBuilder();
        String line = null;
        while ((line = in.readLine()) != null) {
            errorStr.append(line).append("\n");
        }
        return errorStr.toString();
    }

    /**
     * Retorna uma String contendo a saida do console de saida de
     *    um processo (stdout).
     * @param p Objeto Process, que representa um processo.
     * @return Saida do console de saida (stdout).
     * @throws IOException
     */
    public static String getProcessOutStr(Process p) throws IOException {
        BufferedReader in =
            new BufferedReader(new InputStreamReader(p.getInputStream()));
        StringBuilder outStr = new StringBuilder();
        String line = null;
        while ((line = in.readLine()) != null) {
            outStr.append(line).append("\n");
        }
        return outStr.toString();
    }
}
```

5 MANUAL DE INSTALAÇÃO

Esta seção apresenta instruções de configuração e operação da prova de conceito implementada neste trabalho. Algumas dessas instruções se referem a atividades já realizadas previamente, como a implantação dos sistemas e serviços na infraestrutura virtualizada na nuvem, e a preparação dos cartões inteligentes (*smart cards*) com os certificados dos cidadãos fictícios para teste.

5.1 Operação da Prova de Conceito

Aqui estão informações sobre a operação dos sistemas que compõem a prova de conceito, pressupondo que eles já estão implantados e operando corretamente nos servidores virtualizados (conforme explicado na seção 5.2, “Implantação dos Sistemas na Infraestrutura”). Também é pressuposta a existência de cartões inteligentes (*smart cards*) para teste, preparados conforme as seções 5.3, “Emissão de Certificados Digitais para Teste” e 5.4, “Preparação de *Smart Cards* para Teste”.

5.1.1 Sistemas Operacionais e Navegadores *Web* testados

O acesso aos sistemas que compõem a prova de conceito deste trabalho pode ser realizado a partir de qualquer computador que possua conexão à *internet*, um navegador *web* e o *plugin* Java (JRE) devidamente instalado. Entretanto, para usar o leitor de *smart cards*, o sistema operacional deverá ser compatível tanto com o equipamento leitor como também com os cartões utilizados.

Durante o desenvolvimento desta prova de conceito, alguns sistemas operacionais e navegadores *web* foram testados. Eles estão listados a seguir.

Sistema Operacional	Service Pack	Arquitetura
Microsoft Windows XP Professional	SP3	32 bits
Microsoft Windows 2003 Server ([2])	SP2	32 bits
Microsoft Windows Vista	SP2	32 bits e 64 bits
Microsoft Windows 7	SP1	32 bits e 64 bits
Ubuntu Linux 10.04	-	32 bits
Ubuntu Linux 11.04	-	64 bits
Ubuntu Linux 12.04	-	32 bits e 64 bits
CentOS Linux 6	-	32 bits e 64 bits
Fedora Core 13	-	64 bits

Quadro 26 - Sistemas operacionais testados durante o desenvolvimento

Navegador *Web*	Versões Testadas
Microsoft Internet Explorer	9.0
Mozilla Firefox	3.6 / 4.0 / 10.0 / 12.0 / 16.0.2
Google Chrome	18.0 / 23.0

Quadro 27 - Navegadores *web* testados durante o desenvolvimento

Em todos os casos, foi usado o *plugin* Java (JRE) fornecido pela Oracle (ORACLE, 2012), nas versões 6 ou 7 (tanto para *Microsoft Windows* como para *GNU/Linux*, nesse último caso, também nas edições de 64 bits do *plugin*).

[2] O protocolo de criptografia usado nos certificados emitidos pela ICP desta prova de conceito não é compatível com essa versão do *Windows*. Por esse motivo, somente o navegador *web Mozilla Firefox* (que possui infraestrutura própria de gerenciamento de certificados) obteve sucesso no acesso aos sistemas.

5.1.2 Instalação do Leitor de *Smart Cards*

Para ler os certificados digitais armazenados nos cartões de identificação dos cidadãos, é necessário instalar um equipamento leitor de *smart cards* no computador que fará acesso aos sistemas da prova de conceito.

O leitor utilizado durante o desenvolvimento deste trabalho tem conexão USB, e requer a instalação prévia dos *drivers* de dispositivo apropriados. Suas principais especificações são apresentadas a seguir.

Figura 25 - Leitor de *smart cards* SCR3310
Fonte: Identive Group, Product SCR3310 (IDENTIVE, 2012).

O *Identive Infrastructure* SCR3310 é um leitor USB de *smart cards*, compatível com o padrão ISO 7816, e pode ser usado com cartões no formato ID 1 (IDENTIVE, 2012).

Interface	*Full Speed* USB 2.0 (12 Mbps); USB *Powered*
Smart Card	T=0, T=1; ISO 7816; 412 kbps; PC/SC; CT-API
Sistemas Operacionais	Microsoft Windows (98SE, ME, 2000, 2003, 2008, XP, Vista, 7, 8, CE) ; GNU/Linux (mínimo *kernel* 2.4); MacOS; Solaris

Quadro 28 - Especificações do leitor de *smart cards* SCR3310v2.0

Os procedimentos para instalação dos *drivers* do leitor SCR3310 são diferentes para cada sistema operacional suportado. A seguir é descrito o procedimento de instalação para *Microsoft Windows* e *GNU/Linux*.

- **Instalação do *driver* do leitor para *Microsoft Windows***

Algumas versões mais recentes do *Microsoft Windows* possuem no repositório do *Windows Update* um *driver* compatível com o leitor SCR3310. Apesar disso, é recomendável instalar a versão mais atualizada do *driver* fornecido pelo fabricante do *hardware* (IDENTIVE, 2012). Para instalar a mesma versão do *driver* usado durante o desenvolvimento da prova de conceito deste trabalho, pode-se proceder conforme as instruções:

1. Realizar o *download* do pacote de instalação do *driver* do leitor SCR3310, disponível no endereço: <http://tcc.fiap.robsonmartins.com/driver/win/SCR3310_8.51.exe> (acesso em 16 nov. 2012).

2. Executar o arquivo (com permissões de usuário **Administrador**) e seguir as instruções apresentadas, avançando o assistente de instalação até a sua conclusão.

3. Apesar de não obrigatório, é recomendável reiniciar o sistema operacional após a instalação do *driver*.

- **Instalação do *driver* do leitor para *GNU/Linux***

O leitor SCR3310 é suportado pelo pacote *PCSC Lite* (MUSCLE, 2003). O procedimento de instalação desse pacote varia conforme a distribuição *GNU/Linux*, mas a maioria delas já o traz em seus repositórios de pacotes.

As instruções para instalação do pacote *PCSC Lite* no *Ubuntu Linux* (ou outras distribuições derivadas do *Debian*) são as seguintes:

1. Abrir um console (terminal) e digitar o comando abaixo (como usuário **root** - dependendo da configuração, usar o **sudo** antes do comando para ganhar acesso administrativo):

```
# apt-get install libpcsclite1 pcscd pcsc-tools libccid libnss3-tools
```

2. Reiniciar o sistema operacional após a instalação.

As instruções para instalação do pacote *PCSC Lite* no *Red Hat Enterprise Linux* (ou outras distribuições derivadas, como *CentOS* ou *Fedora Core*) são as seguintes:

1. Abrir um console (terminal) e digitar o comando abaixo (como usuário **root** - dependendo da configuração, usar o **sudo** antes do comando para ganhar acesso administrativo):

```
# yum install pcsc-lite pcsc-lite-libs ccid nss
```

2. Reiniciar o sistema operacional após a instalação.

5.1.3 Instalação do *Smart Card*

O *smart card* também é um dispositivo de *hardware*, pois possui um *chip* (circuito integrado). Por esse motivo, ele também necessita da instalação de um *driver* de dispositivo (mais conhecido por *middleware*), fornecido por seu fabricante.

Além disso, o padrão PKCS#11 (ISO/IEC, 1998), usado como meio de acesso ao *smart card* pela tecnologia *Java Cryptography Architecture* (JCA), também necessita de uma biblioteca nativa específica, fornecida pelo fabricante do cartão.

A implementação da *Applet* de Autenticação usada nesta prova de conceito prevê a utilização de somente alguns dispositivos criptográficos (*smart cards* e *tokens*) mais comuns no mercado, conforme listado no código-fonte da enumeração **`SICidApplet/CryptoDeviceType.java`** (descrito na seção 4.1, Código-fonte da *Applet* de Autenticação do Serviço SICid).

O cartão utilizado durante o desenvolvimento deste trabalho é o modelo *IDProtect LASER*, fabricado pela *Athena Smart Card Solutions* (ATHENA, 2012) e fornecido no Brasil pela Pronova Soluções Inteligentes (PRONOVA, 2012). Suas principais especificações são apresentadas a seguir.

Figura 26 - Cartão inteligente (*smart card*) *IDProtect LASER*
Fonte: Pronova Soluções Inteligentes (PRONOVA, 2012).

O *IDProtect LASER* é um *smart card* microprocessado, dotado de um módulo criptográfico e destinado a aplicações como identificação digital, armazenamento de certificados digitais e sistemas de pagamentos (ATHENA, 2012).

Principais características (PRONOVA, 2012):

- Geração *on-board* de par de chaves RSA de até 2048 bits;
- Suporte embutido para os algoritmos criptográficos: RSA, DSA, MD5, SHA1, SHA-2, DES e 3DES;
- Suporte padrão para aplicações *Microsoft Crypto API*;
- Geração de números aleatórios em *hardware*;
- Forte conectividade com aplicações PKI (ICP).
- Assinatura digital realizada em *hardware*;
- Suporte para múltiplos armazenamentos de chaves;
- Certificações CE, FCC e FIPS (FIPS 140);
- Gerenciamento através de senhas PIN e PUK;
- *Middleware* compatível com *Microsoft Windows* e *GNU/Linux*;
- Compatível com Leitor de *Smart Card* T=0 ou T=1.

O fabricante do *IDProtect* (*Athena SCS*) não oferece diretamente o *driver* (*middleware*) para *download*. No entanto, a distribuidora brasileira Pronova fornece um pesado pacote de instalação, que além do *driver*, inclui aplicativos desenvolvidos por ela, para gerenciamento do cartão (alteração de PIN e PUK, inclusão de certificados, formatação, monitoração, etc.). Por esse motivo, nesta prova de conceito, optou-se pelo reempacotamento somente do *driver* básico em um instalador simplificado para cada sistema operacional suportado. O procedimento de instalação do *driver* descrito aqui é específico para utilização desses pacotes de instalação básicos. Caso haja necessidade da instalação dos utilitários de gerenciamento, o pacote oferecido pela Pronova deverá ser usado (como descrito na seção 5.4, “Preparação de *Smart Cards* para Teste”). A seguir é descrito o procedimento de instalação do *driver* básico para *Microsoft Windows* e *GNU/Linux*.

- **Instalação do *driver* do *Smart Card* para *Microsoft Windows***

1. Realizar o *download* do pacote de instalação do *driver* do *smart card IDProtect LASER*, disponível no endereço: <http://tcc.fiap.robsonmartins.com/driver/win/AthenaSCS_6.13.12.exe> (acesso em 16 nov. 2012).

2. Executar o arquivo (com permissões de usuário **Administrador**) e seguir as instruções apresentadas, avançando o assistente de instalação até a sua conclusão.

3. Apesar de não obrigatório, é recomendável reiniciar o sistema operacional após a instalação do *driver*.

- **Instalação do *driver* do *Smart Card* para *GNU/Linux***

1. Realizar o *download* do pacote de instalação do *driver* do *smart card IDProtect LASER*, disponível no endereço: <http://tcc.fiap.robsonmartins.com/driver/linux/AthenaSCS_6.13.12.tar.gz> (acesso em 16 nov. 2012).

2. Abrir um console (terminal) e digitar os comandos abaixo:

```
$ cd <diretório onde está o arquivo AthenaSCS_6.13.12.tar.gz>
$ tar -zxvf AthenaSCS_6.13.12.tar.gz
```

3. Digitar o comando abaixo (como usuário **root** - dependendo da configuração, usar o **sudo** antes do comando para ganhar acesso administrativo):

```
# ./install.sh
```

4. Seguir as instruções apresentadas, até que o *script* de instalação esteja concluído.

5. Apesar de não obrigatório, é recomendável reiniciar o sistema operacional após a instalação do *driver*.

5.1.4 Instalação da Cadeia de Certificados da AC

Para evitar mensagens de "Site não confiável" no navegador *web*, é altamente recomendável a instalação da cadeia de certificados da Autoridade Certificadora emissora dos certificados digitais usados nesta prova de conceito.

Durante a configuração prévia dos sistemas e serviços, foi instituída uma Infraestrutura de Chaves Públicas (ICP) própria para esse fim, a “ICP Robson Martins”. Essa ICP é formada por uma AC Raiz e uma AC Intermediária, que tem, por sua vez, a responsabilidade de emitir os certificados para operação e teste dos sistemas e serviços que compõem a prova de conceito.

A seguir é descrito o procedimento para instalação da cadeia de certificados da “AC Robson Martins” nos navegadores *web* mais populares. Esse mesmo procedimento está publicado e acessível no endereço: <http://tcc.fiap.robsonmartins.com/certbrws/> (acesso em 12 nov. 2012).

- **Navegadores: *Microsoft Internet Explorer*, *Google Chrome* e outros (no *Microsoft Windows*)**

1. Realizar o *download* e salvar o arquivo da “Cadeia de Certificados da Autoridade Certificadora Robson Martins”, que se encontra no endereço: <http://icp.robsonmartins.com/acrmartins.p7b>.

2. Efetuar um clique com o botão direito do mouse sobre o arquivo, selecionando a opção **Instalar Certificado**:

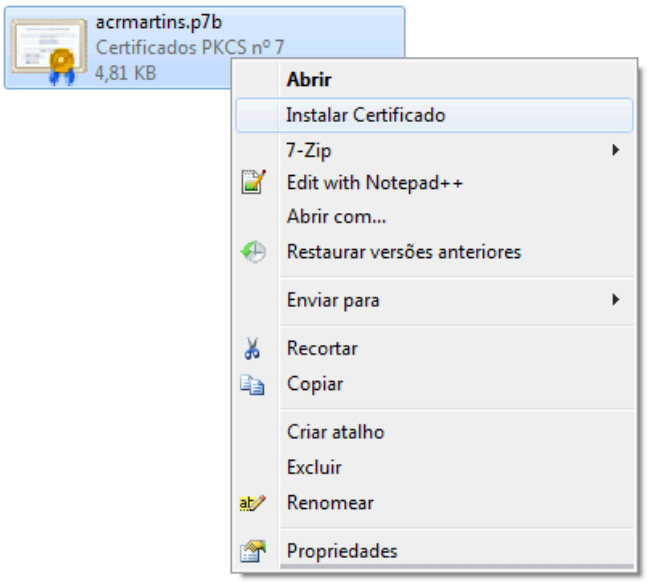

Figura 27 - Instalando cadeia de certificados no *Microsoft Windows*

3. O **Assistente para Importação de Certificados** será inciado. Dar clique sobre o botão **Avançar**:

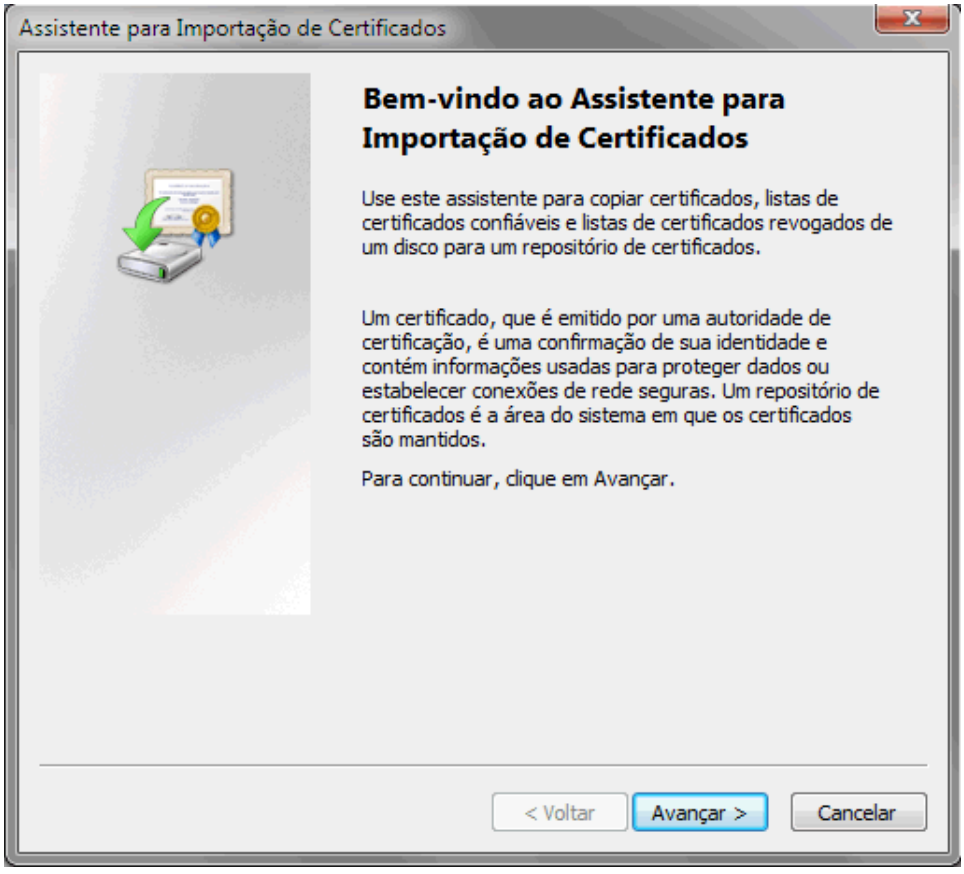

Figura 28 - Assistente para importação de certificados no *Microsoft Windows*

4. Selecionar a opção **Colocar todos os certificados no repositório a seguir**, dar um clique sobre o botão **Procurar...** e selecionar a pasta **Autoridades de Certificação Raiz Confiáveis**. Então, dar um clique sobre o botão **Avançar**:

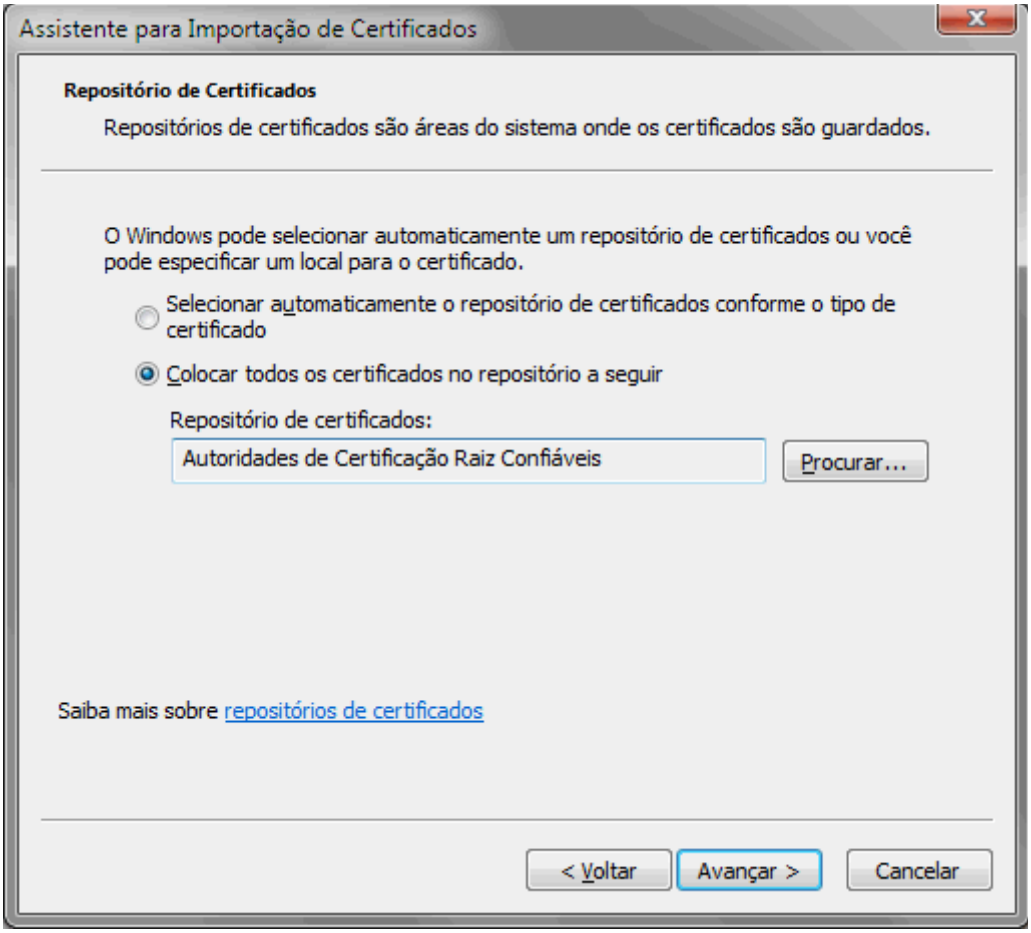

Figura 29 - Seleção de repositório de certificados no *Microsoft Windows*

5. Confirmar a instalação do certificado da Autoridade Certificadora Raiz, através de um clique no botão **Sim**:

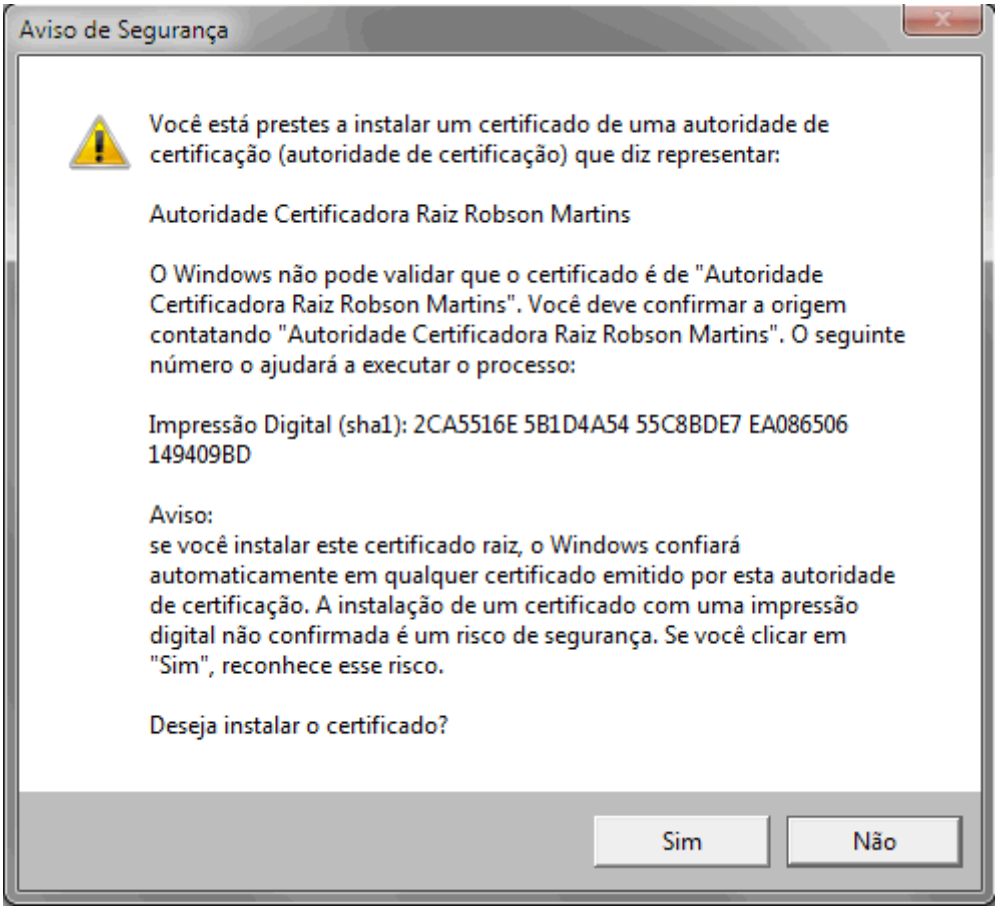

Figura 30 - Instalação do certificado da AC Raiz no *Microsoft Windows*

6. Da mesma forma, confirmar a instalação do certificado da Autoridade Certificadora Intermediária, através de um clique no botão **Sim**.

7. A cadeia de certificados já estará disponível nos navegadores *Microsoft Internet Explorer*, *Google Chrome* e outros (no *Microsoft Windows*).

- **Navegador: *Mozilla Firefox***

1. O *Firefox* só realiza a importação individual dos certificados da cadeia. Realizar o *download* e salvar os arquivos do “Certificado da AC Raiz Robson Martins” e do “Certificado da AC Intermediária Robson Martins”, que se encontram nos endereços: <http://icp.robsonmartins.com/icprmartins.cer> e <http://icp.robsonmartins.com/acrmartins.cer>, respectivamente.

2. No menu do *Firefox*, escolher a opção **Ferramentas**/**Opções** (ou, na versão para *GNU/Linux*, **Editar**/**Preferências**). Selecionar a seção **Avançado**, guia **Criptografia**. Dar um clique sobre o botão **Certificados**:

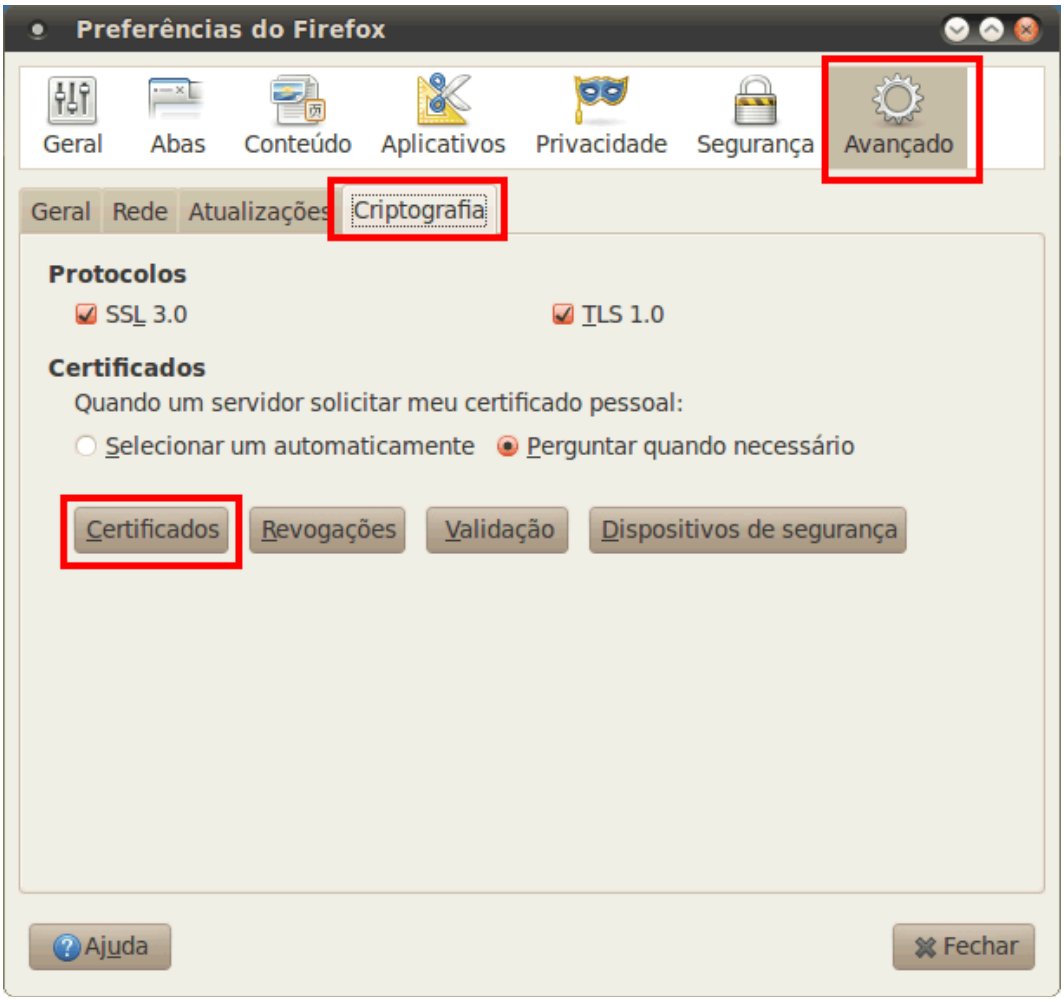

Figura 31 - Preferências de criptografia no *Mozilla Firefox*

3. O **Gerenciador de Certificados** será aberto. Escolher a aba **Autoridades** e dar um clique sobre o botão **Importar...**:

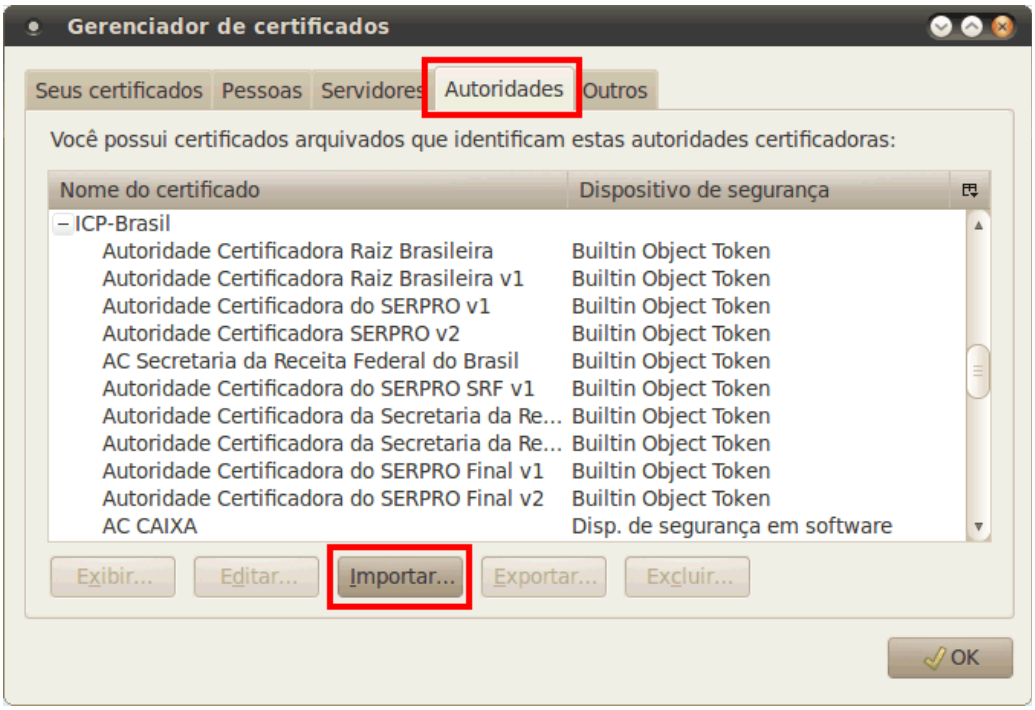

Figura 32 - Gerenciador de certificados do *Mozilla Firefox*

4. Selecionar o arquivo de certificado da AC Raiz (*icprmartins.cer*) e dar um clique sobre o botão **Abrir**:

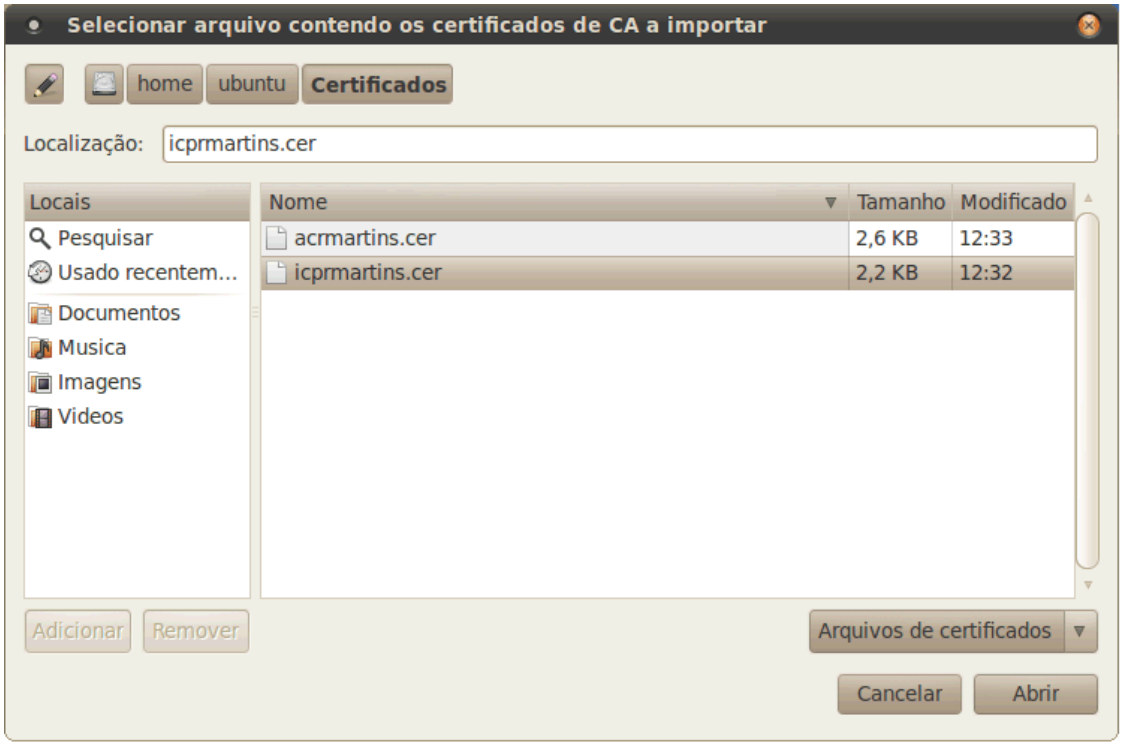

Figura 33 - Diálogo de seleção de arquivo de certificado no *Mozilla Firefox*

5. Confirmar a instalação do certificado da AC Raiz, selecionando todas as opções presentes no diálogo. Dar um clique no botão **OK**:

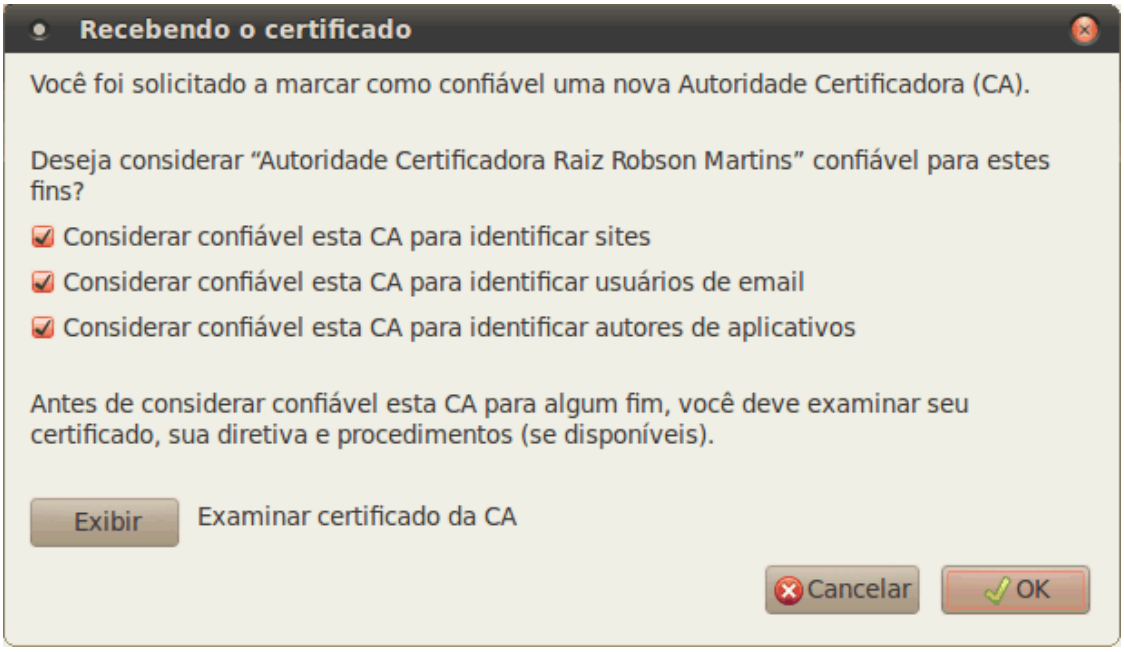

Figura 34 - Instalação do certificado da AC Raiz no *Mozilla Firefox*

6. Semelhantemente, importar o certificado da AC Intermediária (*acrmartins.cer*) e confirmar sua instalação.

7. A cadeia de certificados já estará disponível no navegador *Mozilla Firefox*.

- **Navegador: *Google Chrome* (no *Linux/Unix*)**

1. Realizar o *download* e salvar o arquivo da “Cadeia de Certificados da Autoridade Certificadora Robson Martins”, que se encontra no endereço: <http://icp.robsonmartins.com/acrmartins.p7b>.

2. No menu do *Chrome*, escolher a opção **Configurações...** e selecionar **Configurações Avançadas**. Dar um clique sobre o botão **Gerenciar Certificados**:

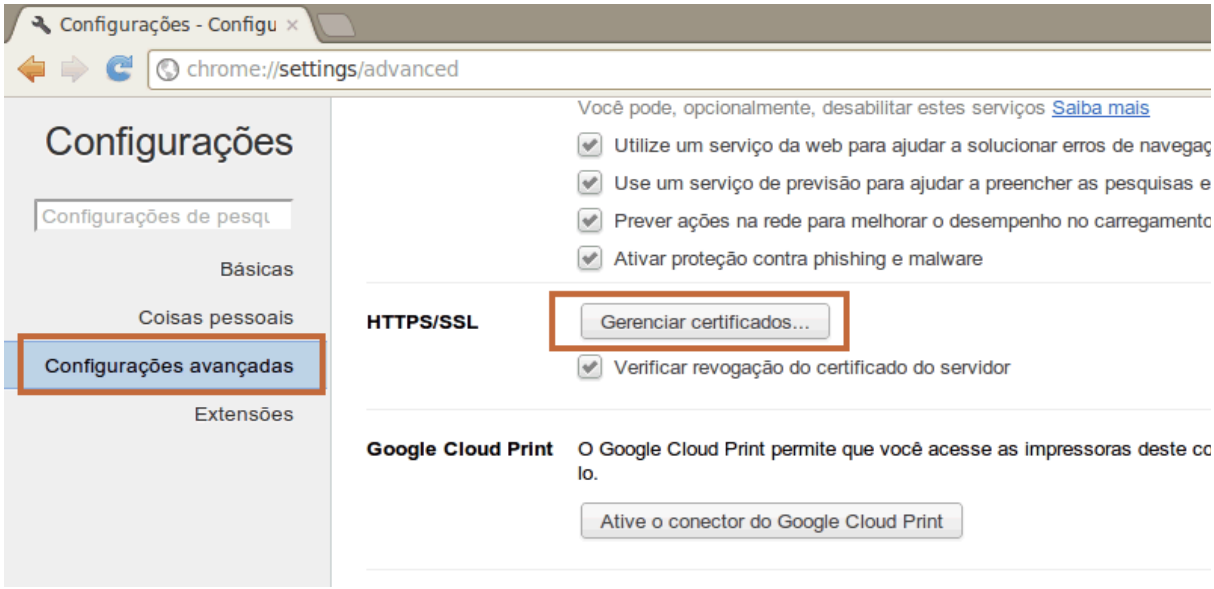

Figura 35 - Configurações avançadas no *Google Chrome*

3. O **Gerenciador de Certificados** será aberto. Escolher a aba **Autoridades** e dar um clique sobre o botão **Importar...**:

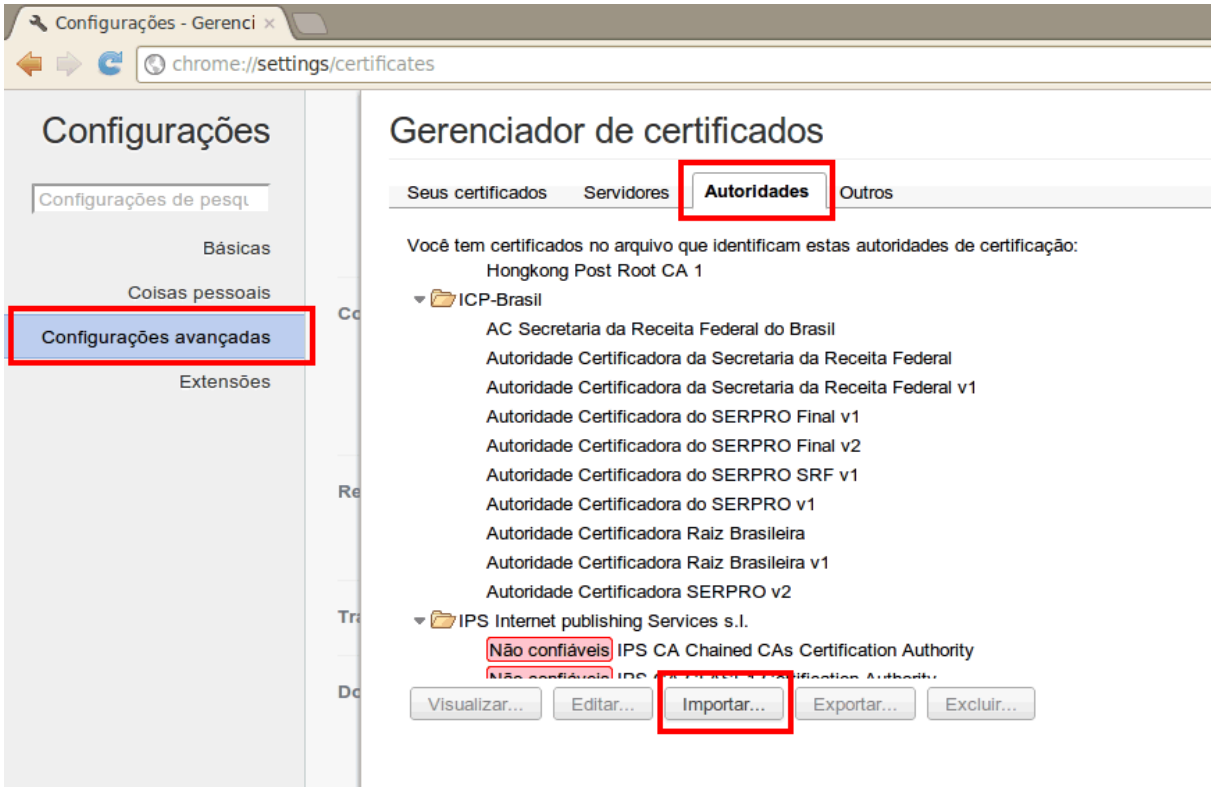

Figura 36 - Gerenciador de certificados do *Google Chrome*

4. Escolher a opção **PKCS #7, cadeia de certificados**, selecionar o arquivo da cadeia (*acrmartins.p7b*) e dar um clique sobre o botão **Abrir**:

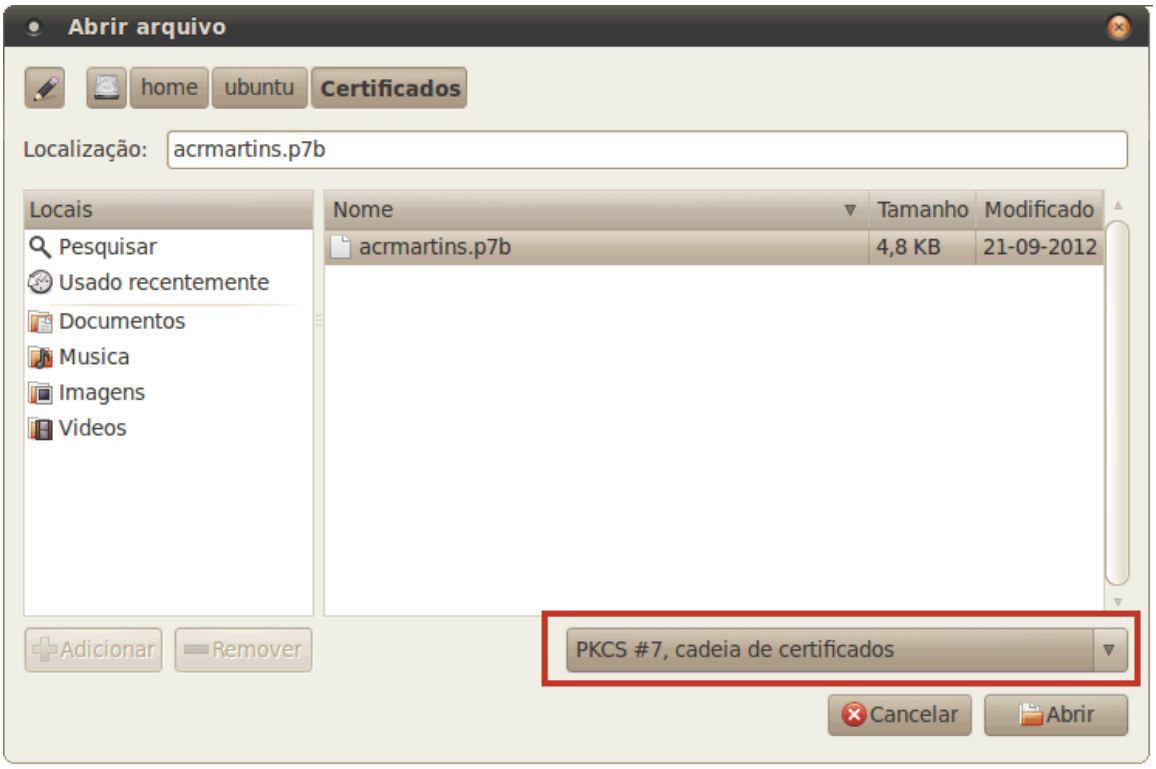

Figura 37 - Diálogo de seleção de arquivo de certificado no *Google Chrome*

5. Confirmar a instalação dos certificados da Autoridade Certificadora, selecionando todas as opções presentes no diálogo. Dar um clique no botão **OK**:

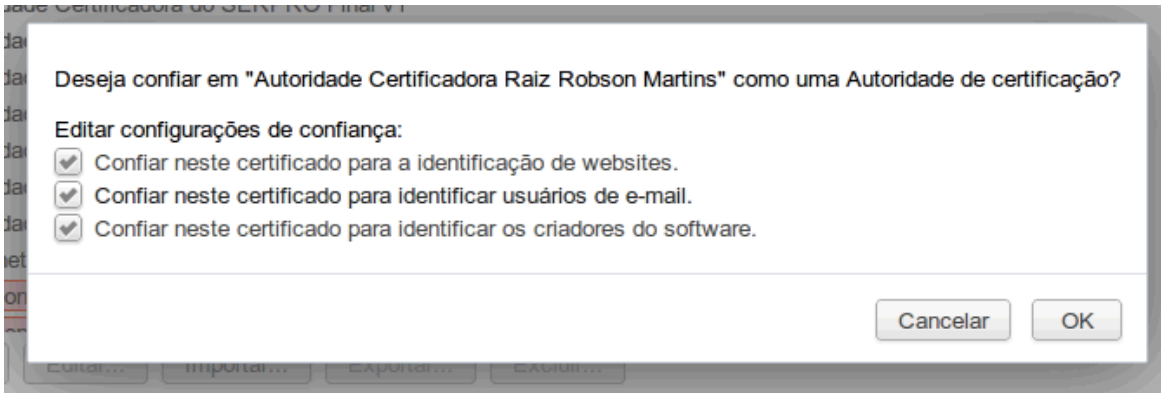

Figura 38 - Instalação do certificado da AC Raiz no *Google Chrome*

6. A cadeia de certificados já estará disponível no navegador *Google Chrome* (no *Linux/Unix*).

5.1.5 Acesso aos Sistemas da Prova de Conceito

O acesso aos sistemas que compõem a prova de conceito deste trabalho pode ser realizado a partir do endereço: <http://tcc.fiap.robsonmartins.com> (acesso em 16 nov. 2012). Esse endereço conduz o navegador a um portal de entrada, que possui além dos *links* (endereços) para os sistemas, algumas informações sobre o projeto.

A autenticação em todos os sistemas é realizada por meio do certificado digital emitido pela Autoridade Certificadora instituída para esse projeto, podendo esse certificado estar armazenado no cartão de identificação (*smart card*), ou então, para efeito de demonstração (sem uso do *hardware* leitor), armazenado em um arquivo protegido (formato PKCS#12).

Para realizar a autenticação, as aplicações *web* utilizam a *applet* de *login*, que deverá ser exibida caso o *plugin* Java esteja corretamente instalado. Na primeira vez que a *applet* for executada, será necessário confirmar os diálogos de segurança apresentados.

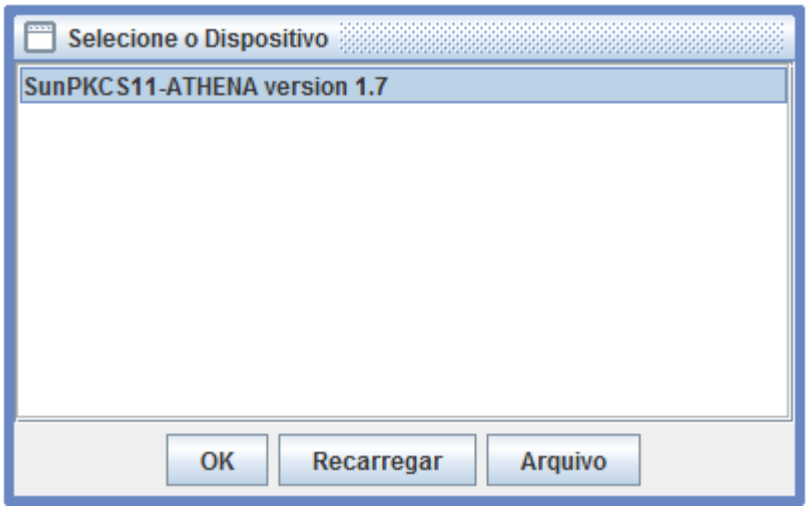

Figura 39 - *Applet* de autenticação nos sistemas da prova de conceito

O procedimento para autenticação (*login*) é o seguinte:

1. Deve-se selecionar o dispositivo criptográfico apresentado na lista e dar um clique no botão **OK**. O botão **Recarregar** serve para listar novamente os dispositivos (após inserir um *smart card* no leitor, por exemplo). Já o botão **Arquivo** possibilita a leitura de um certificado a partir de um arquivo no formato PKCS#12.

2. Após selecionar o dispositivo, deve-se digitar a senha PKCS#11 (PIN) que o protege (ou no caso de utilizar um arquivo PKCS#12, as senhas de *keystore* e do par de chaves – *key password*).

3. Será exibida uma lista com os certificados armazenados no dispositivo (ou no arquivo). Deve-se selecionar o certificado que será usado e dar um clique no botão **OK**. O botão **Voltar** permite retornar novamente à lista de dispositivos criptográficos.

4. O *login* será efetuado e a aplicação poderá ser acessada. Caso haja algum problema de autenticação ou autorização, será exibida uma mensagem de "Certificado inválido".

5.1.6 Administração do Serviço de Identificação do Cidadão (SICid)

O SICid Admin é um sistema que permite a um Administrador gerenciar o Serviço de Identificação do Cidadão (SICid). Seu acesso se dá através do endereço: <https://tcc.fiap.robsonmartins.com/sicid>.

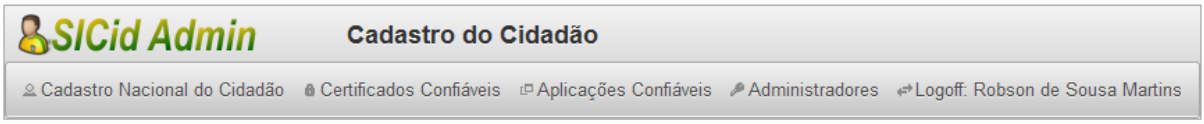

Figura 40 - Menu principal do SICid Admin

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções:

- **Cadastro Nacional do Cidadão**: Permite incluir, excluir ou alterar os dados cadastrais de um cidadão, acessíveis através do Serviço de Identificação do Cidadão.

- **Certificados Confiáveis**: Permite incluir ou excluir os certificados que pertencem a uma cadeia considerada confiável pelo Serviço de Identificação do Cidadão. Caso não haja nenhum certificado neste cadastro, o SICid irá considerar confiáveis quaisquer certificados emitidos por qualquer cadeia.

- **Aplicações Confiáveis**: Permite incluir ou excluir os certificados das aplicações consideradas confiáveis, que podem consumir os serviços oferecidos pelo SICid. Caso não haja nenhuma aplicação cadastrada, o SICid considerará confiável qualquer aplicação que possua um certificado emitido pela cadeia confiável.

- **Administradores**: Permite incluir ou excluir os certificados dos usuários com perfil de Administrador, que podem acessar esta aplicação. Caso não haja nenhum administrador cadastrado, qualquer usuário que possua um certificado emitido pela cadeia confiável poderá acessar a aplicação.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.1.7 Administração do Banco Seguro

O Banco Seguro fornece serviços de *internet banking* a seus clientes. Através de seu sistema, um Gerente pode realizar tarefas administrativas no banco. O acesso à aplicação *web* se dá através do endereço: <https://tcc.fiap.robsonmartins.com/bancoseguro>.

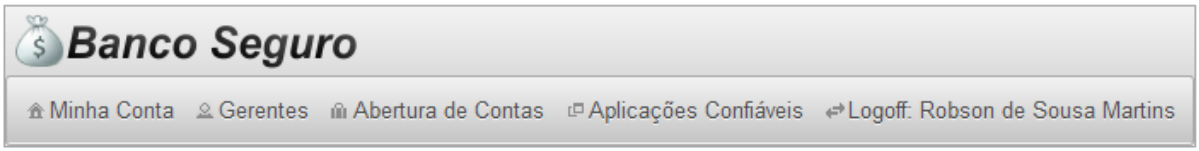

Figura 41 - Menu principal do Banco Seguro – acesso Gerente

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções a um Gerente:

- **Gerentes**: Permite incluir ou excluir os certificados dos usuários com perfil de Gerente, que podem efetuar determinadas tarefas na aplicação. Caso não haja nenhum gerente cadastrado, qualquer usuário que possua um certificado emitido pela cadeia confiável poderá acessar a aplicação como Gerente.

- **Abertura de Contas**: Permite abrir novas contas, ou fechar contas existentes, de clientes do banco.

- **Aplicações Confiáveis**: Permite incluir ou excluir os certificados das aplicações consideradas confiáveis, que podem consumir os serviços oferecidos pelo Banco Seguro. Caso não haja nenhuma aplicação cadastrada, o Banco Seguro considerará confiável qualquer aplicação que possua um certificado emitido pela cadeia confiável.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.1.8 Administração da Receita Nacional

A Receita Nacional é um órgão de governo responsável pela arrecadação periódica de tributos pelos cidadãos. Através de seu sistema, um Administrador (Representante de Governo) pode realizar tarefas administrativas. O acesso à aplicação *web* se dá através do endereço: <https://tcc.fiap.robsonmartins.com/receita>.

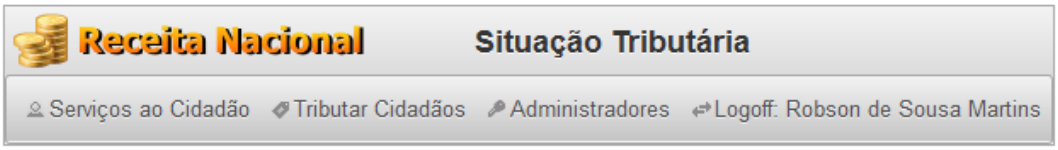

Figura 42 - Menu principal da Receita Nacional – acesso Administrador

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções a um Administrador:

- **Tributar Cidadãos**: Permite calcular e atribuir impostos a todos os cidadãos presentes no Cadastro Nacional do Cidadão.

- **Administradores**: Permite incluir ou excluir os certificados dos usuários com perfil de Administrador, que podem efetuar determinadas tarefas na aplicação. Caso não haja nenhum administrador cadastrado, qualquer usuário que possua um certificado emitido pela cadeia confiável poderá acessar a aplicação como Administrador.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.1.9 Acesso de Cliente ao Banco Seguro

O Banco Seguro fornece serviços de *internet banking* a seus clientes. Através de seu sistema, um Cliente pode realizar a movimentação de sua conta. O acesso à aplicação *web* se dá através do endereço: <https://tcc.fiap.robsonmartins.com/bancoseguro>.

Figura 43 - Menu principal do Banco Seguro – acesso Cliente

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções a um Cliente:

- **Minha Conta**: Exibe o extrato da conta e acesso às operações de movimentação.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.1.10 Acesso de Cidadão à Receita Nacional

A Receita Nacional é um órgão de governo responsável pela arrecadação periódica de tributos pelos cidadãos. Através de seu sistema, um Cidadão pode consultar sua situação tributária e pagar impostos. O acesso à aplicação *web* se dá através do endereço: <https://tcc.fiap.robsonmartins.com/receita>.

Figura 44 - Menu principal da Receita Nacional – acesso Cidadão

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções a um Cidadão:

- **Serviços ao Cidadão**: Exibe a situação tributária e permite pagar um imposto devido.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.2 Implantação dos Sistemas na Infraestrutura

Para realizar a implantação dos sistemas que compõem a prova de conceito nos servidores da infraestrutura, é necessário primeiramente que esses servidores estejam configurados e executando uma instância do servidor de aplicação. O procedimento de preparação dos servidores virtualizados (dentro da infraestrutura de *cloud computing* utilizada neste trabalho) pode ser encontrado no apêndice B, “Implantação na Nuvem”.

A seguir são descritas as instruções para implantação dos sistemas nos servidores a partir de seu código-fonte.

1. Importar o código-fonte (projetos) no ambiente Eclipse, configurando um *server runtime* do JBoss 6.1 para os projetos. Também é necessário utilizar o Oracle Java JDK 7 na configuração de Java VM.

2. Gerar um pacote EAR (*Enterprise Application*) correspondente a cada servidor, a partir da exportação dos projetos *SICidEApp*, *BancoSeguroEApp*, *ReceitaNacionalEApp* e *ICPAdminEApp*.

3. Realizar o *deploy* de cada pacote EAR em seu servidor de aplicação correspondente, copiando-o para o diretório **`<home-server-jboss>/deploy`**.

4. Configurar cada uma das aplicações nos servidores JBoss correspondentes, conforme os procedimentos descritos a seguir.

5.2.1 Configuração do Serviço de Identificação do Cidadão (SICid)

1. Criar o arquivo **`sicid-ds.xml`** no diretório **`<home-server-jboss>/deploy`**. Esse é o arquivo de configuração do *Data Source* para o banco de dados usado pelo SICid (HSQLDB).

sicid-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC
        "-//JBoss//DTD Web Application 5.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>SICidDS</jndi-name>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <connection-url>
      jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}sicidDB
    </connection-url>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

2. Configurar o arquivo **`<home-server-jboss>/conf/login-config.xml`**, adicionando as entradas correspondentes ao *Login Module* JAAS para autenticação das aplicações SICidService e SICidWeb.

login-config.xml

```
<application-policy name="SICidService">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
      <module-option name="dsJndiName">java:/SICidDS</module-option>
      <module-option name="principalsQuery">
        SELECT id as PrincipalID from ConsumidorConfiavel WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' from ConsumidorConfiavel WHERE id=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM ConsumidorConfiavel
      </module-option>
```

login-config.xml

```
      <module-option name="principalIdForEmpty">wsuser</module-option>
      <module-option name="roleForEmpty">wsuser</module-option>
      <module-option name="sicid.disable">true</module-option>
    </login-module>
  </authentication>
</application-policy>

<application-policy name="SICidWeb">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
      <module-option name="dsJndiName">java:/SICidDS</module-option>
      <module-option name="principalsQuery">
        SELECT username as PrincipalID FROM Usuario WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' FROM Usuario WHERE username=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM Usuario
      </module-option>
      <module-option name="principalIdForEmpty">admin</module-option>
      <module-option name="roleForEmpty">admin</module-option>
      <module-option name="sicid.clientProps">sicidweb.properties</module-option>
    </login-module>
  </authentication>
</application-policy>
```

3. Criar o arquivo **`sicidweb.properties`** no diretório **`<home-server-jboss>/conf`**. Esse é o arquivo de configuração de uma aplicação consumidora dos serviços do SICid (nesse caso, a aplicação SICid Admin).

sicidweb.properties

```
sicid.url=https://tcc.fiap.robsonmartins.com/sicid/service/sicid
sicid.namespace=http://ws.sicid/
sicid.service=SICidService
sicid.keyStore=props/sicid.keystore
sicid.storeType=JKS
sicid.storePass=sicid
sicid.keyAlias=sicid
sicid.keyPass=sicid
```

4. Criar o arquivo **`sicid.keystore`** no diretório **`<home-server-jboss>/conf/props`**. Esse é o arquivo de *keystore* que contém o certificado e o par de chaves da aplicação consumidora dos serviços do SICid (nesse caso, a aplicação SICid Admin). A ferramenta Portecle pode ser usada para converter um arquivo PKCS#12 para o formato *Java Key Store* (JKS) requerido.

5.2.2 Configuração do Sistema Banco Seguro

1. Criar o arquivo **`bancoseguro-ds.xml`** no diretório **`<home-server-jboss>/deploy`**. Esse é o arquivo de configuração do *Data Source* para o banco de dados usado pelo Banco Seguro (HSQLDB).

bancoseguro-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC
        "-//JBoss//DTD Web Application 5.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>BancoSeguroDS</jndi-name>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <connection-url>
      jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}bancoseguroDB
    </connection-url>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

2. Configurar o arquivo **`<home-server-jboss>/conf/login-config.xml`**, adicionando as entradas correspondentes ao *Login Module* JAAS para autenticação das aplicações BancoSeguro e BancoSeguroService.

login-config.xml

```
<application-policy name="BancoSeguro">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
```

login-config.xml

```
      <module-option name="dsJndiName">java:/BancoSeguroDS</module-option>
      <module-option name="principalsQuery">
        SELECT cpf as PrincipalID FROM Usuario WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' FROM Usuario WHERE cpf=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM Usuario
      </module-option>
      <module-option name="principalIdForEmpty">gerente</module-option>
      <module-option name="roleForEmpty">gerente</module-option>
      <module-option name="sicid.clientProps">
        bancoseguro.properties
      </module-option>
    </login-module>
  </authentication>
</application-policy>

<application-policy name="BancoSeguroService">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
      <module-option name="dsJndiName">java:/BancoSeguroDS</module-option>
      <module-option name="principalsQuery">
        SELECT id as PrincipalID from ConsumidorConfiavel WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' from ConsumidorConfiavel WHERE id=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM ConsumidorConfiavel
      </module-option>
      <module-option name="principalIdForEmpty">wsuser</module-option>
      <module-option name="roleForEmpty">wsuser</module-option>
      <module-option name="sicid.clientProps">
        bancoseguro.properties
      </module-option>
    </login-module>
  </authentication>
</application-policy>
```

3. Criar o arquivo **bancoseguro.properties** no diretório **<home-server-jboss>/conf**. Esse é o arquivo de configuração de uma aplicação consumidora dos serviços do SICid (nesse caso, a aplicação Banco Seguro).

bancoseguro.properties

```
sicid.url=https://tcc.fiap.robsonmartins.com/sicid/service/sicid
sicid.namespace=http://ws.sicid/
sicid.service=SICidService
sicid.keyStore=props/bancoseguro.keystore
sicid.storeType=JKS
sicid.storePass=banco
sicid.keyAlias=banco
sicid.keyPass=banco
```

4. Criar o arquivo **bancoseguro.keystore** no diretório **<home-server-jboss>/conf/props**. Esse é o arquivo de *keystore* que contém o certificado e o par de chaves da aplicação consumidora dos serviços do SICid (nesse caso, a aplicação Banco Seguro). A ferramenta Portecle pode ser usada para converter um arquivo PKCS#12 para o formato *Java Key Store* (JKS) requerido.

5.2.3 Configuração do Sistema Receita Nacional

1. Criar o arquivo **receita-ds.xml** no diretório **<home-server-jboss>/deploy**. Esse é o arquivo de configuração do *Data Source* para o banco de dados usado pela Receita Nacional (HSQLDB).

receita-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC
        "-//JBoss//DTD Web Application 5.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>ReceitaNacionalDS</jndi-name>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
```

receita-ds.xml

```
    <connection-url>
      jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}receitaDB
    </connection-url>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

2. Configurar o arquivo **<home-server-jboss>/conf/login-config.xml**, adicionando as entradas correspondentes ao *Login Module* JAAS para autenticação da aplicação ReceitaNacional.

login-config.xml

```
<application-policy name="ReceitaNacional">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
      <module-option name="dsJndiName">java:/ReceitaNacionalDS</module-option>
      <module-option name="principalsQuery">
        SELECT ric as PrincipalID FROM Cidadao WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' FROM Cidadao WHERE ric=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM Cidadao WHERE role='governo'
      </module-option>
      <module-option name="principalIdForEmpty">governo</module-option>
      <module-option name="roleForEmpty">governo</module-option>
      <module-option name="sicid.clientProps">receita.properties</module-option>
    </login-module>
  </authentication>
</application-policy>
```

3. Criar o arquivo **`receita.properties`** no diretório **`<home-server-jboss>/conf`**. Esse é o arquivo de configuração de uma aplicação consumidora dos serviços do SICid e do Banco Seguro (nesse caso, a aplicação Receita Nacional).

receita.properties

```
#servico SICid
sicid.url=https://tcc.fiap.robsonmartins.com/sicid/service/sicid
sicid.namespace=http://ws.sicid/
sicid.service=SICidService
sicid.keyStore=props/receita.keystore
sicid.storeType=JKS
sicid.storePass=receita
sicid.keyAlias=receita
sicid.keyPass=receita

#servico Banco Seguro
banco.url=https://tcc.fiap.robsonmartins.com/bancoseguro/service/banco
banco.namespace=http://ws.bancoseguro/
banco.service=BancoSeguroService
banco.keyStore=props/receita.keystore
banco.storeType=JKS
banco.storePass=receita
banco.keyAlias=receita
banco.keyPass=receita
```

4. Criar o arquivo **`receita.keystore`** no diretório **`<home-server-jboss>/conf/props`**. Esse é o arquivo de *keystore* que contém o certificado e o par de chaves da aplicação consumidora dos serviços do SICid e do Banco Seguro (nesse caso, a aplicação Receita Nacional). A ferramenta Portecle pode ser usada para converter um arquivo PKCS#12 para o formato *Java Key Store* (JKS) requerido.

5.2.4 Configuração do Sistema ICP Admin

1. Criar o arquivo **`icpadmin-ds.xml`** no diretório **`<home-server-jboss>/deploy`**. Esse é o arquivo de configuração do *Data Source* para o banco de dados usado pelo ICP Admin (HSQLDB).

icpadmin-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC
        "-//JBoss//DTD Web Application 5.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss-ds_5_0.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>ICPAdminDS</jndi-name>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <connection-url>
      jdbc:hsqldb:${jboss.server.data.dir}${/}icpadmin${/}icpadminDB
    </connection-url>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

2. Configurar o arquivo **`<home-server-jboss>/conf/login-config.xml`**, adicionando as entradas correspondentes ao *Login Module* JAAS para autenticação da aplicação ICP Admin.

login-config.xml

```
<application-policy name="ICPAdmin">
  <authentication>
    <login-module code="com.robsonmartins.fiap.tcc.sicid.jaas.SICidDBLoginModule"
                  flag="required">
      <module-option name="dsJndiName">java:/ICPAdminDS</module-option>
      <module-option name="principalsQuery">
        SELECT username as PrincipalID FROM Usuario WHERE dname=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT role as Role, 'Roles' FROM Usuario WHERE username=?
      </module-option>
      <module-option name="fullPrincipalsQuery">
        SELECT * FROM Usuario
      </module-option>
```

login-config.xml

```
      <module-option name="principalIdForEmpty">admin</module-option>
      <module-option name="roleForEmpty">admin</module-option>
      <module-option name="sicid.clientProps">icpadmin.properties</module-option>
    </login-module>
  </authentication>
</application-policy>
```

3. Criar o arquivo **`icpadmin.properties`** no diretório **`<home-server-jboss>/conf`**. Esse é o arquivo de configuração de uma aplicação consumidora dos serviços do SICid (nesse caso, a aplicação ICP Admin), e além disso, contém outros parâmetros de configuração do ICP Admin.

icpadmin.properties

```
#raiz dos arquivos de dados do ICP Admin (OpenSSL)
icpFilesDir = ${jboss.server.data.dir}${/}icpadmin

#servico SICid
sicid.url=https://tcc.fiap.robsonmartins.com/sicid/service/sicid
sicid.namespace=http://ws.sicid/
sicid.service=SICidService
sicid.keyStore=props/icpadmin.keystore
sicid.storeType=JKS
sicid.storePass=icpadmin
sicid.keyAlias=icpadmin
sicid.keyPass=icpadmin
```

4. Criar o arquivo **`icpadmin.keystore`** no diretório **`<home-server-jboss>/conf/props`**. Esse é o arquivo de *keystore* que contém o certificado e o par de chaves da aplicação consumidora dos serviços do SICid (nesse caso, a aplicação ICP Admin). A ferramenta Portecle pode ser usada para converter um arquivo PKCS#12 para o formato *Java Key Store* (JKS) requerido.

5.3 Emissão de Certificados Digitais para Teste

Os certificados digitais dos servidores, aplicações, serviços e cidadãos fictícios desta prova de conceito foram emitidos através do sistema ICP Admin, que é uma aplicação destinada à administração de uma Infraestrutura de Chaves Públicas (baseada no utilitário *OpenSSL*). Maiores detalhes sobre a cadeia de certificação implementada neste trabalho estão no apêndice A, “Infraestrutura de Chaves Públicas (ICP)”.

As senhas dos certificados emitidos e das Autoridades Certificadoras instituídas neste trabalho são as seguintes:

Certificado	Senha
Autoridade Certificadora Raiz Robson Martins (icprmartins)	icprmartins
Autoridade Certificadora Robson Martins (acrmartins)	acrmartins
Todos certificados emitidos pela AC Robson Martins (até 26/11/2012)	rmartins

Quadro 29 - Senhas dos certificados digitais emitidos para teste

Para emitir novos certificados, revogar ou renovar certificados existentes, ou mesmo criar uma nova cadeia de certificação (Autoridades Certificadoras Raiz e Intermediárias), deve-se utilizar a aplicação *web* ICP Admin, conforme descrito a seguir.

5.3.1 Administração da Infraestrutura de Chaves Públicas

O ICP Admin é um sistema que permite a um Administrador gerenciar todo o ciclo de vida de uma ICP, incluindo criação de Autoridades Certificadoras, emissão, revogação e renovação de certificados, e exportação de lista de certificados revogados, além de outras funções relacionadas. Seu acesso se dá através do endereço: <https://tcc.fiap.robsonmartins.com/icpadmin>.

Figura 45 - Menu principal do ICP Admin

Através dos botões presentes no menu principal da aplicação, estão disponíveis as seguintes funções:

- **AC Raiz**: Permite gerenciar Autoridades Certificadoras Raiz (*Root CA*).

- **AC Intermediária**: Permite gerenciar Autoridades Certificadoras Intermediárias (CA).

- **Certificados**: Permite listar, emitir, revogar, renovar certificados, ou efetuar outras operações relacionadas à emissão de certificados digitais.

- **Administradores**: Permite incluir ou excluir os certificados dos usuários com perfil de Administrador, que podem acessar esta aplicação. Caso não haja nenhum administrador cadastrado, qualquer usuário que possua um certificado emitido pela cadeia confiável poderá acessar a aplicação.

- **Backup ICP**: Permite realizar o *download* de um arquivo compactado, contendo todos os arquivos e diretórios de dados de configuração de uma ICP.

- **Logoff**: Limpa a sessão, forçando a necessidade de uma nova autenticação (*login*) para acesso à aplicação.

5.4 Preparação de *Smart Cards* para Teste

Os *smart cards* utilizados nesta prova de conceito (modelo *IDProtect LASER*, fornecidos pela Pronova Soluções Inteligentes) foram previamente gravados com os certificados digitais emitidos a partir da aplicação ICP Admin, detalhada na seção anterior (5.3.1). Desta forma, para efeito de demonstração, a função de geração de par de chaves criptográficas por *hardware* presente no cartão não foi utilizada.

Como descrito na seção 5.1.3, “Instalação do *Smart Card*”, o pacote de *driver* (*middleware*) usado não contém os utilitários necessários para gerenciar o cartão. Por esse motivo, a instalação do pacote disponibilizado pela Pronova é requerido para a execução dessa tarefa (PRONOVA, 2012).

O procedimento para armazenar um certificado digital no *smart card IDProtect* é descrito a seguir.

1. Realizar o *download* e instalar o gerenciador do cartão *IDProtect LASER*, disponibilizado no endereço: <http://www.pronova.com.br/downloads/smartcards.php> (acesso em 12 nov. 2012).

2. Inserir o *smart card* no leitor (o *driver* do leitor de cartões deverá ter sido previamente instalado).

3. Executar o utilitário **Gerenciador PKI**.

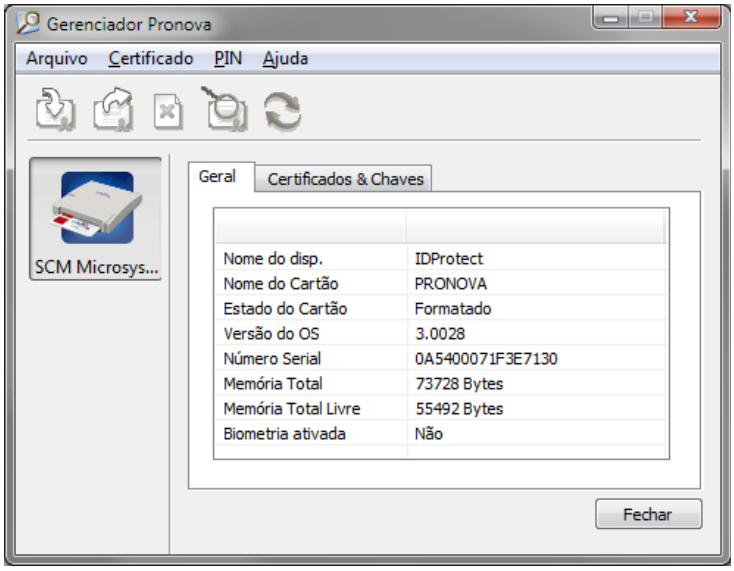

Figura 46 - Gerenciador PKI da Pronova

4. Selecionar o leitor (neste caso é o SCM *Microsystems* SCR3310) e utilizar a aba **Certificados & Chaves** (será solicitada a digitação do PIN que protege o cartão, cujo valor padrão de fábrica é 12345678). Dar um clique sobre o botão **Importar Certificado**.

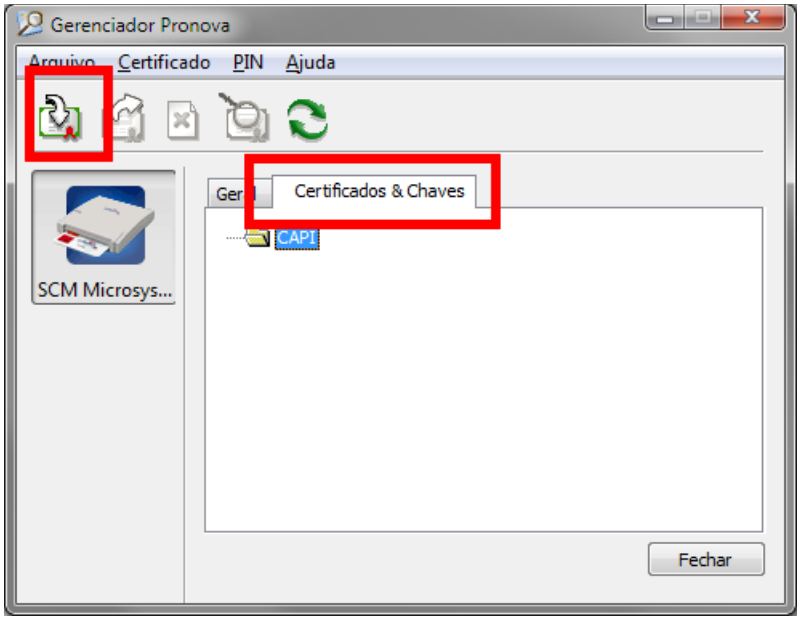

Figura 47 - Gerenciador PKI da Pronova – Certificados & Chaves

5. Deverá ser selecionado o arquivo PFX (no formato PKCS#12), que contém o certificado e o par de chaves a serem importados. Também deve ser especificada a senha que protege o arquivo (*keystore* e *key password* - que devem ser coincidentes nesse caso), e atribuído um nome amigável ao certificado - é recomendado utilizar o valor do campo *Common Name* (CN) para seguir o mesmo padrão adotado pela ICP-Brasil.

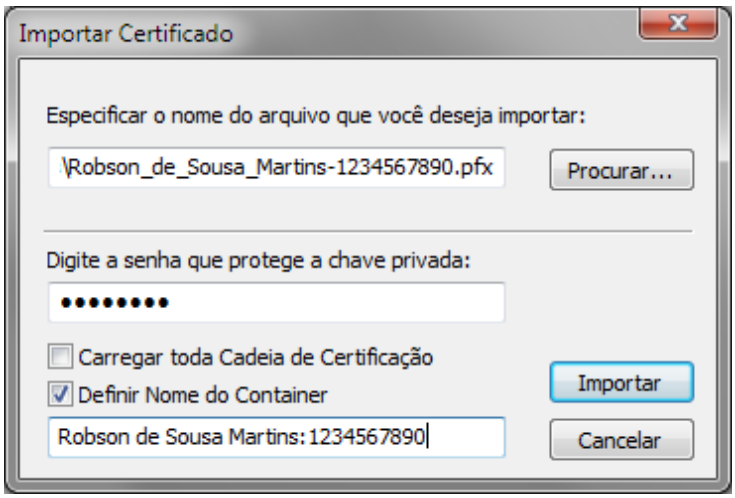

Figura 48 - Gerenciador PKI da Pronova – Importar Certificado

6. Após efetuar um clique no botão **Importar**, o certificado (juntamente com sua chave privada correspondente) estará armazenado dentro do *smart card*.

7. O mesmo **Gerenciador PKI** permite excluir um certificado armazenado em um *smart card*.

A senha que protege o *smart card* (*Personal Identification Number* - PIN) poderá ser inserida incorretamente até um número máximo de vezes, quando então o cartão é bloqueado, e somente a senha *PIN Unlock Key* (PUK) pode desbloqueá-lo. No caso de ultrapassar o limite de tentativas incorretas de digitação do código PUK, o *smart card* é definitivamente inutilizado, e deverá ser descartado.

O valor padrão de ambas as senhas PIN e PUK no *smart card* usado (*IDProtect LASER*) é 12345678. Para alterar as senhas PIN e PUK do *smart card*, devem ser executados os utilitários **Pronova PINTool** e **Pronova Admin PINTool**, respectivamente.

Para formatar o *smart card*, ou seja, apagar todo seu conteúdo, especificar as senhas PIN e PUK e configurar o número máximo de tentativas incorretas de cada uma delas, deve-se usar o utilitário **Formatador Pronova**.

Maiores informações sobre os utilitários de gerenciamento da Pronova podem ser obtidas no Manual de Usuário disponibilizado junto com as ferramentas (PRONOVA, 2012).

6 CONCLUSÃO

Este trabalho apresentou alguns exemplos de arquitetura de integração de sistemas, com o objetivo de simplificar a interação entre o cidadão e o governo. De maneira não-exaustiva, as ideias foram expostas como ponto de partida para o desenvolvimento de projetos reais dentro dessa temática. Além das vantagens e benefícios, também foram salientados alguns dos principais pontos críticos na implementação e implantação de cada abordagem.

O desenvolvimento de uma prova de conceito serviu como experimentação real das ideias apresentadas, especialmente demonstrando a integração de sistemas com o emprego da Arquitetura Orientada a Serviços (SOA), em conjunto com algumas Tecnologias de Informação e Comunicação (TIC) já existentes.

O uso de tecnologias Java no desenvolvimento da prova de conceito, apesar de não ser essencial para demonstrar as ideias aqui apresentadas, simplificou sobremaneira a elaboração deste trabalho, pois muitos dos padrões tecnológicos empregados são completamente implementados pela arquitetura padrão Java (tais como *web services*, criptografia assimétrica, assinatura digital, autenticação e autorização), ou então por bibliotecas oferecidas nesta linguagem.

Além disso, a implantação dos sistemas e serviços que compõem a prova de conceito em servidores virtualizados numa infraestrutura em nuvem, demonstrou de forma prática como uma plataforma *cloud computing* pode ser benéfica se associada com a ideia de uma Arquitetura Orientada a Serviços (SOA), pois a própria infraestrutura de servidores e rede se torna um serviço utilizável pelas aplicações.

Alguns aspectos de segurança também foram demonstrados através da implementação da prova de conceito (apesar de não ser o foco principal deste trabalho), tais como a necessidade de criptografia, autenticação e comprovação de identidade através de assinatura digital.

Em suma, a consequência desse trabalho foi a demonstração de como serviços de alto valor ao cidadão podem ser disponibilizados, oferecendo simplicidade de acesso através do uso de um cartão único de identificação.

Na próxima seção são apresentadas algumas ideias correlatas a este trabalho, que poderão servir como base a futuras pesquisas e ao desenvolvimento de outros trabalhos, derivados a partir da temática abordada neste.

6.1 Trabalhos Derivados

A identificação única digital e a integração de sistemas podem servir como base a uma série de outros trabalhos correlatos, todos compartilhando o propósito de simplificar a vida do cidadão nas suas relações com o governo, com as empresas e com a sociedade de uma maneira geral.

Uma das possibilidades seria a utilização do cartão de identificação (como o RIC), através de tecnologia *contactless* (comunicação sem contatos), no pagamento de passagens no transporte público, nacionalmente, substituindo os diversos cartões empregados atualmente nas grandes cidades (e incompatíveis de cidade para cidade).

Como uma ideia para aumentar a segurança do cartão de identificação, uma possibilidade seria a inclusão de elementos biométricos digitalizados em seu *chip* (como impressão digital ou imagem da íris). Assim, um leitor de cartões poderia também validar a identificação do cidadão através de biometria.

Dentro da mesma ideia de biometria, unida à integração de sistemas, uma aplicação de segurança pública poderia utilizar computadores de mão (*tablets*) para reconhecer impressões digitais e ler cartões de identificação (como o RIC). Desta forma, um agente policial poderia verificar a identidade de um cidadão suspeito durante uma abordagem, e em tempo real obter sua ficha criminal.

Outra abordagem seria a substituição do cartão de identificação (como o RIC) por um *chip* presente no *smartphone* do cidadão, incluindo também segurança através de biometria, no caso de roubo ou perda do equipamento. Assim, o aparelho poderia ser usado automaticamente como vetor de autenticação do usuário em transações (bancárias, com empresas ou com o governo), eliminando até mesmo a necessidade de um computador para acesso aos serviços.

Enfim, as tecnologias de informação e comunicação atuais já permitem a elaboração de diversos projetos com o objetivo de beneficiar o cidadão. Basta a aplicação prática dessas tecnologias para que a vida do cidadão moderno se torne cada vez mais simplificada. E a do governo e das empresas, mais eficiente.

REFERÊNCIAS

ACM SIGKDD - Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining. **Data Mining Curriculum**. 2006. Disponível em: <http://www.sigkdd.org/curriculum.php>. Acesso em: 12 nov. 2012.

AMAZON. **Amazon Elastic Compute Cloud (Amazon EC2)**. 2012. Disponível em: <http://aws.amazon.com/pt/ec2/>. Acesso em: 12 nov. 2012.

APACHE Software Foundation. **Apache Ant**. 2003. Disponível em: <http://ant. apache.org/>. Acesso em: 12 nov. 2012.

_____. **Apache Commons**. 2012. Disponível em: <http://commons.apache.org>. Acesso em: 12 nov. 2012.

_____. **Apache HTTP Server**. 1996. Disponível em: <http://httpd.apache.org/>. Acesso em: 12 nov. 2012.

_____. **Log4J:** Logging Services for Java. 2004. Disponível em: <http://logging. apache.org/>. Acesso em: 12 nov. 2012.

ATHENA Smart Card Solutions Inc. **Athena-SCS web site**. 2012. Disponível em: <http://www.athena-scs.com/>. Acesso em: 12 nov. 2012.

BARRY, D. K.; G ANNON , P. J. **Web services and service-oriented architecture**: The savvy manager's guide. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

BOUNCY CASTLE, The Legion Of The. **Java cryptography APIs**. 2012. Disponível em: <http://www.bouncycastle.org/java.html>. Acesso em: 12 nov. 2012.

BRASIL, Instituto Nacional de Tecnologia da Informação. **Portal do ITI**. 2011. Disponível em: <http://www.iti.gov.br>. Acesso em: 31 jul. 2012.

BRASIL, Ministério da Justiça. **Registro de Identidade Civil**. 2010. Disponível em: <http://portal.mj.gov.br/ric>. Acesso em: 09 jan. 2012.

_____. **Resolução nº 2, de 10 de setembro de 2010**. Diário Oficial da União da República Federativa do Brasil. Brasília, DF, 27 set. 2010, Seção 1, p. 31.

BRASIL, Ministério do Planejamento, Orçamento e Gestão. **e-PING – Padrões de Interoperabilidade do Governo Eletrônico**. 2012. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/ e-ping-padroes-de-interoperabilidade>. Acesso em: 31 jul. 2012.

BRASIL, Palácio do Planalto. **Medida Provisória 2200-2, de 24 de agosto de 2001**. Publicada no Diário Oficial Eletrônico. Brasília, DF, 27 ago. 2001, p. 65. Disponível em: <http://legislacao.planalto.gov.br/legisla/legislacao.nsf/Viw_Identificacao/ mpv 2.200-2-2001>. Acesso em: 31 jul. 2012.

BRASIL, Receita Federal do Brasil – Ministério da Fazenda. **Orientações Gerais - Emissão, Renovação e Revogação de Certificados Digitais e-CPF ou e-CNPJ**. 2011. Disponível em: <http://www.receita.fazenda.gov.br/atendvirtual/SolicEmRenRevCD.htm>. Acesso em: 31 jul. 2012.

BRASIL, Secretaria da Receita Federal – Ministério da Fazenda. **Instrução Normativa SRF nº 156, de 22 de dezembro de 1999**. Diário Oficial da União da República Federativa do Brasil. Brasília, DF, 27 dez. 1999, p. 23-35.

BRASIL, Serviço Federal de Processamento de Dados. Integrar para crescer. **Revista Tema**. Brasília, 2003. Ano 28, edição 169, set./out. 2003.

_____. Democracia 3.0. **Revista Tema**. Brasília, 2011. Ano 37, edição 205, abr. 2011.

CANONICAL. **Ubuntu Linux**. 2012. Disponível em: <http://www.ubuntu.com/>. Acesso em: 12 nov. 2012.

CERTSIGN Certificadora Digital S.A. **Certificado e-CNPJ**. 2011. Disponível em: <http://www.certisign.com.br/produtos-e-servicos/certificados-digitais/e-cnpj>. Acesso em: 31 jul. 2012.

_____. **Certificado e-CPF**. 2011. Disponível em: <http://www.certisign.com.br/produtos-e-servicos/certificados-digitais/e-cpf>. Acesso em: 31 jul. 2012.

CHAPPELL, David. **Enterprise Service Bus**: Theory in Practice. USA: O'Reilly Media, 2004. ISBN: 978-0-59-600675-4.

DAVIE, Bruce S.; PETERSON, Larry L. **Computer Networks ISE**: A Systems Approach. 4 ed. USA: Elsevier, 2007. ISBN: 978-01-2374-013-7.

DRAEGER, Scott. **Is it Buzz or Best Practice? Unraveling the Mystery of Service Oriented Architecture**. 2008. Disponível em: <http://www.energypulse.net/centers/article/article_display.cfm?a_id=1704>. Acesso em: 31 jul. 2012.

ECLIPSE FOUNDATION. **The Eclipse Project**. 2012. Disponível em: <http://www.eclipse.org>. Acesso em: 12 nov. 2012.

ERL, Thomas. **Service-Oriented Architecture (SOA)**: Concepts, Technology and Design. USA: Prentice Hall, 2005.

FERREIRA, Aurélio. B. H. *Aurélio século XXI: o dicionário da Língua Portuguesa*. 3. ed. rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

FIELDING, Roy T. **Architectural Styles and the Design of Network-based Software Architectures**. Dissertação (Doctor Of Philosophy) - Information And Computer Science, University Of California, Irvine, 2000. cap. 5.

GALLIERS, R. D.; SUTHERLAND, A. R. **Information systems management and strategy formulation:** the 'stages of growth' model revisited. *Journal of Information Systems*. 1991. Volume 1, Issue 2, p. 89–114.

GIMP - The GIMP Team. **GNU Image Manipulation Program**. 2001. Disponível em: <http://www.gimp.org/>. Acesso em: 12 nov. 2012.

GUILHEN, Stefan N. **Um serviço de autorização Java EE baseado em certificados de atributos X.509**. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2008.

GUIMARÃES, Romero Wanderley. **Investigação de uma Arquitetura de Sistemas de Informação para o Governo de Pernambuco**. Dissertação (Mestrado em Ciência da Computação) - Centro de Informática, Universidade Federal de Pernambuco, Pernambuco. 2009.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. 3 ed. Porto Alegre: Sagra Luzzatto, 2000. ISBN: 978-85-241-0590-9.

HSQL Development Group. **HyperSQL DB**. 2012. Disponível em: <http://hsqldb.org> . Acesso em: 12 nov. 2012.

IBM. **Rational Team Concert**. 2008. Disponível em: <https://jazz.net/products/ rational-team-concert/>. Acesso em: 12 nov. 2012.

IDENTIVE Group. **SCR3310/v2 Smart Card Reader**. 2012. Disponível em: <http://www.identive-infrastructure.com/en/products-solutions/smart-card-readers-a-terminals/smart-card-readers/scr3310>. Acesso em: 14 nov. 2012.

INI4J Project. **The Java ini library**. 2012. Disponível em: <http://ini4j.sourceforge. net/>. Acesso em: 12 nov. 2012.

ISO/IEC 7816. 1998, Identification cards - Integrated circuit(s) cards with contacts.

JAMAE, Javid; JOHNSON, Peter. **JBoss in Action**: Configuring the JBoss Application Server. Greenwich: Manning, 2009. ISBN: 978-1-933988-02-3.

JBOSS Community. **JBoss Application Server**. 2012. Disponível em: <http://www.jboss.org/jbossas>. Acesso em: 12 nov. 2012.

LARMAN, Craig. **Utilizando UML e Padrões**: Uma introdução à análise e ao projeto orientados a objetos. 3 ed. Porto Alegre: Bookman, 2008. ISBN: 978-85-6003-152-8.

LEALDINE, Regiane de Cássia. **Tecnologias da Informação nas Empresas:** por que a TI é importante para as empresas?. 2007. Disponível em: <http://www.webartigos.com/articles/1832/1/tecnologias-da-informacao-nas-empresas/pagina1.html>. Acesso em: 09 jan. 2012.

LINTHICUM, David S. **Cloud Computing and SOA Convergence in Your Enterprise**: a Step-by-Step Guide. Boston: Pearson Education, 2010. ISBN: 978-0-13-600922-1.

LUCKOW, Décio H.; MELO, Alexandre A. de. **Programação Java para a Web**. São Paulo: Novatec Editora, 2010. ISBN: 978-85-7522-238-6.

LUZ, Clarissa P. da. **Centro de Certificação Digital**: Construção, Administração e Manutenção. Rio de Janeiro: Ciência Moderna, 2008. ISBN: 978-85-7393-6933.

MARKS, Eric A.; BELL, Michael. **Service-Oriented Architecture (SOA)**: A Planning and Implementation Guide for Business and Technology. USA: Hardcover, 2006.

MOZILLA. **Firefox Web Browser**. 1998. Disponível em: <http://br.mozdev.org/>. Acesso em: 12 nov. 2012.

MUSCLE - Movement for the Use of Smart Cards in a Linux Environment. **PCSC Lite Project**. 2003. Disponível em: <http://pcsclite.alioth.debian.org/>. Acesso em: 12 nov. 2012.

OASIS Committee Specification. **OASIS Reference Model for Service Oriented Architecture 1.0**. 2006. Disponível em: <https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm>. Acesso em: 31 jul. 2012.

OPENSSL Project. **OpenSSL**. 2009. Disponível em: <http://www.openssl.org/>. Acesso em: 12 nov. 2012.

ORACLE. **Java Platform**. 2012. Disponível em: <http://www.oracle.com/us/technologies/java>. Acesso em: 12 nov. 2012.

_____. **Oracle VM VirtualBox**. 2008. Disponível em: <http://www.virtualbox.org> . Acesso em: 12 nov. 2012.

PORTECLE. **The Portecle Tool**. 2004. Disponível em: <http://portecle.sourceforge.net/>. Acesso em: 12 nov. 2012.

PRIMEFACES. **Ultimate JSF Component Suite**. 2011. Disponível em: <http://primefaces.org/>. Acesso em: 12 nov. 2012.

PRIMEFACES EXTENSIONS. **Additional JSF 2 components for PrimeFaces**. 2012. Disponível em: <http://code.google.com/p/primefaces-extensions/>. Acesso em: 12 nov. 2012.

PRONOVA Soluções Inteligentes. **Produtos para segurança digital**. 2012. Disponível em: <http://www.pronova.com.br/>. Acesso em: 12 nov. 2012.

PULIER, Eric; TAYLOR, Hugh. **Understanding Enterprise SOA**. USA: Manning Publications, 2005.

RANKL, Wolfgang; EFFING, Wolfgang. **Smart Card Handbook**. 3 ed. Translated by Kenneth Cox. Chichester: John Wiley & Sons Ltd, 2003. ISBN 0-470-85668-8.

ROCH, Eric. **SOA Benefits, Challenges and Risk Mitigation**. 2006. Disponível em: <http://it.toolbox.com/blogs/the-soa-blog/soa-benefits-hallenges-and-riskmitigation-8075>. Acesso em: 31 jul. 2012.

RSA Laboratories. **PKCS #7: Cryptographic Message Syntax Standard**. V1.5. 1993. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2129>. Acesso em: 12 nov. 2012.

_____. **PKCS #10: Certification Request Syntax Standard**. V1.7. 2000. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2132>. Acesso em: 12 nov. 2012.

_____. **PKCS #11: Cryptographic Token Interface Standard**. V2.30. 2009. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2133>. Acesso em: 12 nov. 2012.

_____. **PKCS #12: Personal Information Exchange Syntax Standard**. 1999. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2138>. Acesso em: 12 nov. 2012.

SILVA, Luiz Gustavo. et al. **Certificação Digital**: Conceitos e Aplicações. Rio de Janeiro: Ciência Moderna, 2008. ISBN: 978-85-7393-655-1.

SMARTBEAR Software. **The soapUI Functional Testing Tool**. 2005. Disponível em: <http://www.soapui.org/>. Acesso em: 12 nov. 2012.

TDF - The Document Foundation. **The LibreOffice Project**. 2008. Disponível em: <http://pt-br.libreoffice.org/>. Acesso em: 12 nov. 2012.

TRAIN, Sheila. **Identidade digital**: Como e por que os certificados digitais estão facilitando a vida. 2 ed. São Paulo: Certisign Certificadora Digital, 2007. ISBN 978-85-6018-9007.

VELTE, Anthony T. et al. **Cloud Computing - Computação em Nuvem:** Uma Abordagem Prática. Alta Books, 2010. ISBN: 978-85-7608-536-2.

VOLPI, Marlon Marcelo. **Assinatura Digital**: Aspectos Técnicos, Práticos e Legais. Rio de Janeiro: Axcel Books, 2001. ISBN: 85-7323-151-3.

W3C Consortium. **Simple Object Access Protocol (SOAP)**. 2007. Disponível em: <http://www.w3.org/TR/soap/>. Acesso em: 31 jul. 2012.

_____. **Web Services Architecture Requirements**. 2004. Disponível em: <http://www.w3.org/TR/wsa-reqs>. Acesso em: 31 jul. 2012.

_____. **Web Services Description Language (WSDL)**. 2001. Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em: 31 jul. 2012.

WINDLEY, Phillip J. **Digital identity**. USA: O'Reilly, 2005. ISBN 978-0-596-00878-9.

WOOLF, Bobby; HOSKINS, Jim W. **Exploring IBM SOA Technology & Practice**. USA: Clear Horizon, 2008.

YWORKS. **yEd Graph Editor**. 2012. Disponível em: <http://www.yworks.com/en/products_yed_about.html>. Acesso em: 12 nov. 2012.

GLOSSÁRIO

AC: O mesmo que **Autoridade Certificadora**.

AC Intermediária: O mesmo que **Autoridade Certificadora Intermediária**.

AC Raiz: O mesmo que **Autoridade Certificadora Raiz**.

Algoritmo de Resumo: O mesmo que **Digest Algorithm**.

ANS: Agência Nacional de Saúde Suplementar.

API: *Application Programming Interface*. É um conjunto de padrões estabelecidos por um *software* para a utilização das suas funcionalidades por outros aplicativos.

Applet (Java): O mesmo que **Java Applet**.

AR: O mesmo que **Autoridade de Registro**.

Assinatura Digital: É um mecanismo digital utilizado para fornecer confiabilidade, tanto sobre autenticidade de um determinado documento eletrônico, como sobre o remetente do mesmo (VOLPI, 2001).

Autenticação: Verificação da identidade de um determinado ator em um sistema de *software* (seja ele humano ou outro sistema computacional).

Autoridade Certificadora: É uma entidade (ou organização) responsável por emitir certificados digitais. Ela deve ser reconhecidamente confiável, para que os certificados emitidos por ela também possam ser considerados confiáveis e autênticos.

Autoridade Certificadora Intermediária: É uma Autoridade Certificadora cujo certificado digital foi emitido por outra AC, ou seja, ela não tem o maior nível na hierarquia da cadeia de certificados.

Autoridade Certificadora Raiz: É uma Autoridade Certificadora de maior nível na hierarquia da cadeia de certificados, ou seja, seu certificado não é emitido por nenhuma outra AC, é um certificado autoassinado.

Autoridade de Registro: É uma entidade responsável por confirmar, através de análise cadastral e auditoria, a identidade real de um solicitante de emissão de certificado digital, para garantir a veracidade e autenticidade das informações por ele fornecidas.

Autorização: É um mecanismo responsável por garantir que apenas usuários autorizados acessem recursos protegidos de um sistema de *software*.

Base64: É um algoritmo para codificação de dados, com o objetivo de transformar valores binários em texto plano, para posterior processamento em sistemas que lidam apenas com mensagens de texto (caracteres imprimíveis).

BASIC (HTTP): Em uma requisição HTTP, é um método usado para que um cliente forneça um nome de usuário e senha com o objetivo de se autenticar em um servidor *web*. Esses dois parâmetros são enviados no cabeçalho da requisição HTTP de forma clara (sem criptografia).

CA: *Certification Authority*. O mesmo que **Autoridade Certificadora**.

Cadeia de Certificados: É um conjunto que representa uma hierarquia de certificados digitais, a partir do certificado da AC Raiz até o certificado emitido pela última AC Intermediária.

Cartão Inteligente: O mesmo que **Smart Card**.

CEI: Cadastro Específico do **INSS**.

Certificado Autoassinado: É um certificado digital que não é assinado por uma AC de nível superior, mas que possui uma assinatura criada com a própria chave privada correspondente ao certificado.

Certificado Digital: É um arquivo que contém uma chave pública, além de alguns atributos relacionados a entidade (indivíduo ou organização) titular dessa chave, associado a uma assinatura digital emitida por uma autoridade certificadora reconhecida. O certificado garante que o dono do par de chaves criptográficas é realmente quem ele afirma ser (SILVA et al., 2008).

Certificado Digital A1: É um certificado digital fornecido em um arquivo protegido por senha, geralmente tem uma validade inferior ao certificado A3.

Certificado Digital A3: É um certificado digital fornecido em uma mídia criptográfica, como um *smart card* ou um *token*. Como a proteção desse certificado é garantida por *hardware*, sua validade costuma ser maior do que um certificado A1.

Chave Privada: É uma chave criptográfica conhecida somente pelo seu proprietário, e não deve, em hipótese alguma, ser acessível por outros indivíduos. Ela forma par com uma chave pública correspondente, cujo uso faz parte da chamada criptografia assimétrica.

Chave Pública: É uma chave criptográfica que forma par com a chave privada na criptografia assimétrica, mas ao contrário da última, pode ser publicada e conhecida por outros indivíduos. A chave pública é um componente de um certificado digital.

CLIENT-CERT (HTTP): Em uma requisição HTTP, é um método usado para que um cliente forneça um certificado digital com o objetivo de trocar informações com um servidor *web*. É um método usado durante a fase de negociação de protocolos criptográficos de transporte, como o SSL. Por esse motivo, só faz sentido com o uso do protocolo HTTPS, no qual o servidor *web* também possui um certificado digital.

Cloud Computing: É uma forma de computação distribuída, onde aplicações de *software* são executadas em diferentes servidores fisicamente separados, mas logicamente reunidas como um serviço ("*as a service*").

Cluster: É uma forma de computação distribuída, onde vários computadores interligados em rede são reunidos para executar uma determinada aplicação. Tipicamente tem por objetivo aumento de desempenho, aumento de disponibilidade (redundância) ou balanceamento de carga.

CN: O mesmo que **Common Name**.

CNPJ: Cadastro Nacional de Pessoa Jurídica. Identifica empresas e outras organizações e entidades.

Common Name: É um nome usado para identificar um indivíduo ou computador específico num serviço de diretório (padrão LDAP). É aplicado ao padrão X.509 de certificados digitais para identificar o requerente do certificado.

Computação em Nuvem: O mesmo que **Cloud Computing**.

CPF: Cadastro de Pessoa Física. Identifica cidadãos como contribuintes (isentos ou não) na Receita Federal do Brasil (BRASIL, 1999).

Criptografia: É uma escrita secreta, em cifra, isto é, por meio de abreviaturas ou sinais convencionais (FERREIRA, 1999). A criptografia em sistemas computacionais consiste da aplicação de algoritmos matemáticos e de uma chave criptográfica, de forma a tornar uma mensagem digital ilegível. Somente o destinatário da mensagem poderá recuperá-la (torná-la legível novamente), através do mesmo algoritmo e da chave criptográfica correspondente.

Criptografia Assimétrica: Forma de criptografia que utiliza um par de chaves, uma privada e outra pública, através dos quais uma mensagem cifrada com uma das chaves só pode ser revelada por meio da outra chave do mesmo par.

Criptografia Simétrica: Forma de criptografia que usa uma chave para cifrar uma mensagem e a mesma chave para tornar a mensagem legível novamente.

CRL: *Certificate Revocation List*. O mesmo que **Lista de Revogação de Certificados**.

DAO: *Data Access Object* (Objeto para Acesso a Dados). É um padrão para persistência de dados que permite separar as regras de negócio das regras de acesso a banco de dados.

Data Mining: O principal propósito do processo de mineração de dados é extrair informações a partir de um conjunto de dados e transformá-las em uma estrutura compreensível para uso posterior (ACM-SIGKDD, 2006, tradução nossa).

Datasource: É um termo normalmente empregado à configuração de conexão para um servidor (sistema gerenciador) de banco de dados.

Dataware House: É um sistema utilizado para armazenar informações originalmente dispersas em bancos de dados, em uma forma consolidada.

Deploy: É o nome dado ao processo de disponibilizar (ou publicar) uma aplicação de *software* (tipicamente em um servidor *web* ou servidor de aplicações).

DETRAN: Departamento (Estadual) de Trânsito.

Digest Algorithm: Forma de criptografia que transforma uma mensagem plana em uma chave de tamanho fixo (conhecida como *hash*).

Distinguished Name: É usado para identificar uma entrada de forma não ambígua num serviço de diretório (padrão LDAP). É aplicado ao padrão X.509 de certificados digitais para identificar unicamente um certificado.

DN: O mesmo que **Distinguished Name**.

DNS: *Domain Name System* (Sistema de Nomes de Domínio). É um sistema de gerenciamento de nomes hierárquico e distribuído, responsável por traduzir nomes amigáveis de domínio para endereços de *internet* (IP), e vice-versa.

Driver (de dispositivo): É uma porção de *software* que atua nas camadas inferiores do sistema operacional, responsável pela comunicação entre um equipamento de *hardware* e as aplicações de *software* que interagem com esse equipamento.

e-CNPJ: Cadastro Nacional de Pessoa Jurídica (Eletrônico). É uma versão eletrônica do CNPJ, usado por empresas e outras entidades para garantir a autenticidade e a integridade nas suas transações (CERTSIGN, 2011).

e-Commerce: Comércio Eletrônico. É uma modalidade de transação comercial realizada através de um equipamento eletrônico, tipicamente um computador conectado à *internet*.

e-CPF: Cadastro de Pessoa Física (Eletrônico). É uma versão eletrônica do CPF, que garante a autenticidade e a integridade nas transações eletrônicas de pessoas físicas (CERTSIGN, 2011).

e-PING: Padrões de Interoperabilidade de Governo Eletrônico. Define um conjunto mínimo de premissas, políticas e especificações técnicas que regulamentam a utilização da Tecnologia de Informação e Comunicação (TIC) no Governo Federal, estabelecendo as condições de interação com os demais Poderes e esferas de governo e com a sociedade em geral (BRASIL, 2012).

EAR: *Enterprise Archive*. É um formato de arquivo usado na arquitetura Java EE para empacotar um ou mais módulos que formam uma aplicação corporativa, de forma a simplificar o *deploy* de todos os módulos em um único arquivo.

EJB: *Enterprise Java Beans*. É um componente de *software* que faz parte da especificação Java EE, e que essencialmente executa num contêiner (servidor) de aplicação (ORACLE, 2012).

Entity Bean: É um tipo de EJB que representa um dado persistido em banco de dados.

ESB: *Enterprise Service Bus* (Barramento de Serviços). É uma camada de *software* (conhecida como *middleware*), que visa oferecer uma infraestrutura flexível de conectividade para a integração de serviços e aplicações (CHAPPELL, 2004).

FIAP: Faculdade de Informática e Administração Paulista.

Filesystem: Sistema de Arquivos. Refere-se à estrutura lógica de arquivos e diretórios persistida nos dispositivos de armazenamento disponíveis em um sistema computacional.

FORM (HTTP): Em uma requisição HTTP, é um método usado para que um cliente forneça um nome de usuário e senha com o objetivo de se autenticar em um servidor *web*. Esses dois parâmetros são enviados através de um formulário HTML, dentro do corpo da requisição, geralmente usando o método HTTP POST.

Framework: Um conjunto reusável de *software* que provê funcionalidades úteis no desenvolvimento de aplicações computacionais.

GUID: *Globally Unique Identifier*. É um número identificador único para um determinado tipo de sistema computacional.

Hash: Chave de tamanho fixo criada a partir da aplicação de um algoritmo criptográfico de resumo (*digest algorithm*) sobre uma mensagem plana.

Hibernate: Um *framework* destinado à persistência de objetos Java em bancos de dados (JBOSS, 2012). É uma das implementações mais conhecidas e utilizadas de JPA.

HTTP: *Hyper Text Transfer Protocol*. É um protocolo de aplicação, relacionado à *internet*, que permite a transferência de informações entre sistemas através de uma rede de computadores.

HTTPS: *Secure Hyper Text Transfer Protocol*. É o protocolo HTTP usado em conjunto com criptografia, fornecida por outro protocolo como o SSL.

ICP: Infraestrutura de Chaves Públicas. É uma estrutura baseada em uma cadeia de certificados, associada a uma série de controles de segurança (físicos e lógicos), cuja hierarquia garante a confiança e a confiabilidade nas transações baseadas em certificados digitais.

ICP-Brasil: Infraestrutura de Chaves Públicas Brasileira. É uma cadeia hierárquica e de confiança que viabiliza a emissão de certificados digitais para identificação do cidadão (brasileiro) quando transacionando no meio virtual, como a *Internet* (BRASIL, 2011).

INPI: Instituto Nacional de Propriedade Industrial.

INSS: Instituto Nacional do Seguro Social.

Intermediate CA: O mesmo que **Autoridade Certificadora Intermediária**.

Internet Banking: Caracteriza uma aplicação bancária, que oferece serviços de movimentação de contas (e outras transações financeiras) através da *internet*.

ITI: Instituto de Tecnologia da Informação.

JAAS: *Java Authentication and Authorization Service*. É uma API Java que oferece serviços de autenticação e autorização a aplicações construídas no padrão JEE (ORACLE, 2012).

Java: É uma plataforma que oferece recursos para o desenvolvimento de aplicações computacionais (*software*), e compreende uma série de elementos, como linguagem de programação, bibliotecas, *frameworks*, especificações, padrões de arquitetura, etc.

Java Applet: Código baseado em Java (miniaplicativo) capaz de ser executado num computador cliente, num contexto do navegador de *internet*.

Java Enterprise Edition: Plataforma para desenvolvimento de aplicações corporativas usando a arquitetura Java (ORACLE, 2012).

Java Keystore: É um repositório para armazenamento de certificado digitais e chaves criptográficas, num formato de arquivo compatível com a plataforma Java.

Java Plugin: É uma extensão para o navegador de *internet* (*web browser*) que adiciona suporte a execução de miniaplicativos (*applets*), desenvolvidos em Java. É disponibilizado como parte do JRE.

Java Swing: *Framework* gráfico (*widget toolkit*), baseado em Java, que permite a elaboração de interfaces gráficas com o usuário (GUI).

JAXB: *Java Architecture for XML Binding*. É a especificação de uma API que permite a representação de objetos Java como elementos XML e vice-versa (ORACLE, 2012).

JAX-WS: *Java API for XML Web Services*. É a especificação de uma API para a construção de *web services* usando Java (ORACLE, 2012).

JCA: *Java Cryptography Architecture*. Especificação de uma arquitetura destinada a oferecer funcionalidades criptográficas (incluindo manipulação de chaves, assinatura e certificação digital) a aplicações baseadas em Java (ORACLE, 2012).

JCE: *Java Cryptography Extension*. *Framework* que implementa a arquitetura JCA, incluído na distribuição Java (JSE) padrão.

JDK: *Java Development Kit*. É o kit de desenvolvimento de aplicações baseadas em Java. Contém além do JSE padrão, utilitários e ferramentas para desenvolvimento.

JEE: O mesmo que **Java Enterprise Edition**.

JKS: O mesmo que **Java Keystore**.

JPA: *Java Persistence API*. É a especificação de uma API que permite a persistência de objetos Java em bancos de dados (ORACLE, 2012).

JRE: *Java Runtime Environment*. Contém o ambiente de execução para aplicações baseadas em Java, compreendendo a JVM.

JSE: *Java Standard Edition*. A plataforma *Java Standard Edition* (Java SE) possibilita o desenvolvimento e implantação de aplicações baseadas em Java (ORACLE, 2012).

JSF: *Java Server Faces*. Especificação e *framework*, destinados a simplificar a construção de aplicações *web* baseadas em Java.

JSON: *Javascript Object Notation*. É um subconjunto da notação de objetos do JavaScript, que representa estruturas de dados usando mensagens de texto plano.

JSP: *Java Server Pages*. É uma tecnologia usada no desenvolvimento de aplicações *web* em Java, focada na elaboração de páginas dinâmicas processadas no servidor de aplicação.

JVM: *Java Virtual Machine*. É a Máquina Virtual Java, responsável pela interpretação e execução do *bytecode* gerado durante a compilação das aplicações baseadas em Java.

Key Password: Senha que protege uma chave privada, especialmente em arquivos de *keystore* (JKS ou PKCS#12).

Keystore: É um repositório para armazenamento de certificado digitais e chaves criptográficas. Pode ser um arquivo (como o JKS ou o PKCS#12), ou estar contido dentro de um dispositivo criptográfico (*smart card* ou *token*).

Keystore Password: Senha que protege um repositório de *keystore* (tipicamente um arquivo).

LCR: O mesmo que **Lista de Revogação de Certificados**.

LDAP: *Lightweight Directory Access Protocol*. É um protocolo para atualizar e pesquisar diretórios, através de um serviço sendo executado sobre TCP/IP.

Leitor de Cartões Inteligentes: O mesmo que **Leitor de Smart Cards**.

Leitor de Smart Cards: Equipamento capaz de transferir dados entre um *smart card* e um computador, através da conexão em uma porta de comunicação apropriada (como USB).

Lista de Revogação de Certificados: É uma lista que contém os números seriais dos certificados digitais que foram revogados pela Autoridade Certificadora.

Local EJB: É um EJB que define uma interface que permite acesso ao mesmo computador onde ele está sendo executado. Ao contrário, o acesso remoto (via rede), é permitido a um EJB que defina uma interface remota.

Login: Representa a ação de um usuário (ou sistema) ao realizar o processo requerido para autenticação (e posterior autorização) em uma aplicação computacional.

Login Module: Módulo de *software* (desenvolvido sob as especificações do JAAS), que tem por objetivo servir como intermediário no processo de autenticação e autorização em uma aplicação Java EE baseada no JAAS.

Logoff: Processo que reverte um *login*, ou seja, desfaz a autenticação e a autorização do usuário corrente (através da limpeza da sessão). Após o *logoff*, um novo *login* é requerido para acesso aos recursos protegidos de uma aplicação.

Managed Bean: Se refere a um objeto gerenciado pelo contêiner de um servidor de aplicação Java EE (seu ciclo de vida é gerenciado automaticamente pelo contêiner, e não manualmente pela aplicação).

Mineração de Dados: O mesmo que **Data Mining**.

MPOG: Ministério do Planejamento, Orçamento e Gestão.

MVC: *Model-View-Controller*. É um padrão de arquitetura para desenvolvimento de *software*, onde as camadas de modelo (dados e lógica de negócio), controle (mediador) e visualização (interface com usuário) são distintas.

NF-e: Nota Fiscal Eletrônica. É um documento digital, emitido e armazenado eletronicamente, que tem o objetivo de documentar operações de circulação de mercadorias ou prestação de serviços.

OASIS: *Organization for the Advancement of Structured Information Standards*. É um consórcio global que conduz o desenvolvimento, convergência e adoção de padrões de interoperabilidade para troca de informações.

Online: Refere-se a uma atividade executada de forma conectada, em que uma das partes interage com a outra independente da sua distância física.

OpenSSL: Um conjunto de ferramentas (*open-source*) que implementam utilidades criptográficas (OPENSSL, 2009).

PASEP: Programa de Formação do Patrimônio do Servidor Público.

PFX: Formato de arquivo de *keystore* que se refere ao padrão **PKCS#12**.

PIN: *Personal Identification Number*. É uma senha que protege um dispositivo de armazenamento criptográfico (como um *token* ou um *smart card*).

PIS: Programa de Integração Social.

PKCS: *Public Key Cryptography Standards*. Conjunto de padrões de criptografia estabelecidos e publicados pela *RSA Data Security*, uma empresa de segurança computacional.

PKCS#7: Padrão usado para disseminar certificados digitais gerados em uma PKI (RSA, 1993). Arquivos que seguem este padrão têm a extensão P7B, e podem conter um ou mais certificados.

PKCS#11: Padrão relativo a dispositivos criptográficos de *hardware* - tais como *tokens* e *smart cards* (RSA, 2009).

PKCS#12: Padrão relativo a troca de informações sensíveis, tais como chaves privadas (RSA, 1999). Arquivos que seguem este padrão têm a extensão P12 ou PFX, são conhecidos como arquivos de *keystore*, e são protegidos por senha.

PKI: *Public Key Infrastructure*. O mesmo que **ICP**.

Plugin: É uma pequena porção específica de *software* responsável por estender as funcionalidades de uma aplicação (também conhecido como módulo de extensão).

POJO: *Plain Old Java Object*. É um objeto Java que segue um desenho simplificado. É um termo bastante usado como sinônimo de JavaBean, ou seja, uma classe que possui apenas construtor *default* e métodos de acesso (*getters* e *setters* - que obrigatoriamente seguem um padrão definido de nomeação).

Portal: É um *site* na *internet*, que funciona como centro aglomerador e distribuidor de conteúdo para uma série de outros *sites* ou *subsites*.

Proxy: É um servidor (ou serviço) intermediário, que atende a requisições, mas as repassa a outro servidor ou serviço (que é quem processa e responde de fato às requisições).

ProUni: Programa Universidade para Todos.

PUK: *PIN Unlock Key*. É uma senha que desbloqueia um dispositivo de armazenamento criptográfico após um bloqueio provocado por um número de tentativas inválidas de inserção do PIN. Se a senha PUK for inserida incorretamente acima de um número máximo de tentativas, o dispositivo é definitivamente bloqueado e fica inutilizado.

REST: *Representational State Transfer*. É um estilo de arquitetura de *software* para sistemas distribuídos através de protocolos *internet* (como HTTP).

Revogação de Certificado: É o ato de cancelar a validade de um certificado digital, realizado por uma AC, seja por solicitação do requerente (por exemplo, se a senha da chave privada foi roubada por um atacante, o certificado ficou comprometido), ou por revogação do certificado de uma AC participante da hierarquia da cadeia de certificados do emitente (todos os certificados abaixo na hierarquia são obrigatoriamente revogados).

RFB: Receita Federal do Brasil.

RIC: Registro de Identidade Civil. É uma inciativa do Ministério da Justiça que tem por objetivo unificar a identificação dos cidadãos brasileiros em um cadastro de nível nacional, prevendo a utilização de um cartão (*smart card*) para reunir todas as informações sobre o cidadão (BRASIL, 2010).

RG: Registro Geral. Documento de identidade do cidadão brasileiro que possui um número (sem garantias de unicidade), e emitido em caráter Estadual (e não em nível Nacional).

Root CA: O mesmo que **Autoridade Certificadora Raiz**.

RSA: Pode se referir a empresa de segurança computacional *RSA Data Security*, responsável por diversos padrões criptográficos (como o PKCS), ou ao algoritmo de criptografia de mesmo nome, que provê funcionalidades para assinatura digital e criptografia assimétrica.

SAML: *Security Assertion Markup Language*. É um padrão baseado em XML que define um formato para troca de informações de autenticação e autorização entre dois pares.

Servidor de Nomes de Domínio: O mesmo que **DNS**.

Servidor Proxy: O mesmo que **Proxy**.

SHA1: *Secure Hash Algorithm*-1. É um algoritmo de resumo (*digest algorithm*), capaz de transformar um conjunto de dados em um bloco de 160 bits (*hash*), praticamente único para cada conjunto de dados distinto.

Smart Card: É um cartão de plástico, com dimensões semelhantes a um cartão magnético comum, dotado de um *chip* capaz de armazenar de dados sensíveis, chaves criptográficas e certificados digitais (RANKL, 2003).

Smart Card Reader: O mesmo que **Leitor de Smart Cards**.

SOA: *Service-Oriented Architecture* (Arquitetura Orientada a Serviços). É um paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários (OASIS, 2006).

SOAP: *Simple Object Access Protocol*. É um protocolo baseado em XML para troca de informações estruturadas em uma plataforma descentralizada e distribuída (é normalmente usado como base para construção de *web services*).

SPED: Sistema Público de Escrituração Digital.

SSL: *Secure Sockets Layer*. É um protocolo que utiliza criptografia na camada de transporte da rede, visando garantir a confidencialidade das informações (DAVIE, PETERSON, 2007).

Stateless EJB: É um tipo de EJB que não mantém estado durante uma sessão.

Subject Alternative Name: É um conjunto de campos em um certificado digital, que contém as chamadas extensões (atributos estendidos), conforme o padrão X.509.

Subject Name: É o conjunto de campos em um certificado digital que identifica o requerente do certificado.

SUS: Sistema Único de Saúde.

Swing (Java): O mesmo que **Java Swing**.

Token: É um dispositivo portátil que funciona como mídia armazenadora. Em seu *chip* é inserida a chave privada do usuário. O acesso às informações é feito por meio de uma senha pessoal (PIN). O *token* assemelha-se a uma pequena chave (*pendrive*) que é colocada em uma entrada do computador (BRASIL, 2011).

URI: *Uniform Resource Identifier*. É um identificador usado para nomear um recurso de *internet* (comumente chamado de endereço *web*).

USB: *Universal Serial Bus*. É uma porta de comunicação usada para a conexão de periféricos a um computador, frequentemente encontrada nos equipamentos atuais.

VPN: *Virtual Private Network* (Rede Privada Virtual). É uma rede de comunicações privada, construída sobre uma rede pública (como por exemplo, a *Internet*). Normalmente é baseada em "túneis" criptografados que usam protocolos seguros (SSL, por exemplo).

W3C: *World Wide Web Consortium*. É um consórcio internacional que visa a padronização das aplicações na *web*.

Web: É um serviço de acesso à informação, através de uma rede de alcance mundial, podendo usar multimídia (textos, sons, vídeos e imagens), baseado principalmente no protocolo HTTP. Também conhecido como *Internet*.

Web Service: É uma aplicação de software identificada por uma URI, onde interfaces são definidas, descritas e publicadas como artefatos XML. Um *Web Service* permite interações diretas com outros agentes de software por meio da troca de mensagens XML, trafegadas sobre protocolos baseados em *internet* (W3C, 2004, tradução nossa).

WSDL: *Web Services Definition Language*. É uma linguagem baseada em XML que descreve *web services*, na forma de um contrato de serviço.

WS-Security: *Web Services Security*. É uma extensão do SOAP para aplicar segurança a *web services*, através do uso de criptografia e assinatura digital. Faz parte de um conjunto de especificações de segurança para *web services* recomendadas pela OASIS.

WWW: *World Wide Web.* O mesmo que **Web**.

X.509: É um padrão para Infraestrutura de Chaves Públicas (ICP). Ele especifica, dentre vários itens, o formato (estrutura) dos certificados digitais. A versão correntemente usada pelo padrão ICP-Brasil é a v3 (X.509.3).

XML: *eXtensible Markup Language*. É uma linguagem de marcação extensível, baseada em arquivos texto estruturados, capaz de descrever diversos tipos de dados. Seu principal propósito é facilitar a transferência de informações entre sistemas heterogêneos.

XSD: *XML Schema Definition*. É um padrão baseado no formato XML para definição de regras de validação de dados em documentos XML.

APÊNDICE A – INFRAESTRUTURA DE CHAVES PÚBLICAS (ICP)

A Infraestrutura de Chaves Públicas (ICP) é uma estrutura baseada em uma cadeia de certificados, associada a uma série de controles de segurança (físicos e lógicos), cuja hierarquia garante a confiança e a confiabilidade nas transações baseadas em certificados digitais.

Como a prova de conceito que demonstra este trabalho se baseia fortemente no uso de certificados digitais, a existência de uma ICP responsável pela emissão, revogação e renovação de certificados digitais se torna obrigatória.

Devido ao alto custo de se recorrer a Autoridades Certificadoras que pertencem a uma ICP amplamente reconhecida (como a ICP-Brasil, ou entidades internacionais como a *Verisign*), para a emissão de certificados de teste, optou-se, para fins de demonstração, pela instituição de uma ICP própria (com certificado raiz autoassinado).

Para facilitar as tarefas de emissão, revogação e renovação de certificados digitais (que são tarefas de administração do ciclo de vida de uma ICP), foi idealizada uma aplicação *web*, com o objetivo de agregar todas essas funções em uma interface simplificada com o usuário. Essa aplicação é chamada de **ICP Admin**.

A seguir serão apresentados alguns detalhes sobre o desenvolvimento do sistema ICP Admin.

Sistema para Administração da Infraestrutura de Chaves Públicas (ICP Admin):

O ICP Admin é uma aplicação *web* escrita em Java, e é executado em um servidor de aplicação JBoss implantado na infraestrutura virtualizada na nuvem, assim como os outros sistemas que compõem a prova de conceito.

Uma das maiores dificuldades enfrentadas no desenvolvimento dessa aplicação foi encontrar o suporte necessário para manipular certificados digitais, listas de revogação (LCR), conversão entre formatos de arquivo de certificados e *keystore*, etc.

As funções criptográficas disponibilizadas pelo Java (através de JCA e JCE) têm seu escopo bem definido, não sendo suficientes para a realização de todas as tarefas necessárias a administração de uma ICP.

Por esse motivo, uma pesquisa foi realizada em busca de bibliotecas ou *frameworks* que oferecessem esse suporte, sendo então encontradas duas alternativas possíveis:

- ***Bouncy Castle* - Biblioteca criptográfica para Java**: É uma biblioteca nativa escrita em Java que oferece funções criptográficas, incluindo suporte a certificados X.509, padrões PKCS, etc.

- ***OpenSSL***: É um utilitário *open-source* de linha de comando, que provê diversas funções criptográficas, relacionadas a certificados e assinatura digital, protocolos SSL, padrões PKCS, etc.

A biblioteca *Bouncy Castle*, por ser nativa em Java, mostrou-se a melhor opção para oferecer o suporte necessário. No entanto, após estudo da sua documentação e de exemplos reais de implementação, chegou-se a conclusão de que a complexidade de código seria muito grande para desenvolver, num espaço de tempo tão curto para a elaboração deste trabalho, todas as tarefas necessárias.

Então, como o utilitário *OpenSSL* possui todo o suporte necessário, e sua interface se faz por meio de simples arquivos de configuração, diretórios e argumentos de linha de comando, ele foi escolhido para oferecer as funcionalidades principais dessa aplicação.

Mediante essa decisão, alguns passos foram tomados para materializar esse trabalho:

- Foi elaborado um estudo dentro da documentação (*man pages*) do *OpenSSL*, de forma a obter quais os parâmetros de configuração e argumentos de linha de comando necessários para a realização de cada tarefa (funcionalidade) do ICP Admin.

- Foram desenhadas e implementadas classes *wrapper* para as chamadas de sistema que se referem ao *OpenSSL* e à manipulação de arquivos e diretórios. Todas as chamadas remotas (via aplicação *web*) aos métodos implementados foram protegidas contra execução concorrente (usando *synchronized*), para evitar erros provocados por uma descoordenação na ordem das chamadas de sistema relacionadas.

- Uma biblioteca Java (Ini4j) foi utilizada para dar suporte facilitado à manipulação dos arquivos de configuração do *OpenSSL* (que têm um formato texto semelhante aos arquivos INI do *Microsoft Windows*).

Formatos dos certificados emitidos pela ICP-Brasil:

Para que os certificados emitidos através do sistema ICP Admin ficassem semelhantes aos emitidos pela ICP-Brasil (com os mesmos tipos, atributos e formatos), houve um trabalho de pesquisa dos padrões da ICP-Brasil, e sua posterior implementação no desenvolvimento das funções para emissão de certificados do ICP Admin.

ICP Robson Martins:

Uma vez que o sistema ICP Admin foi implantado, uma ICP própria foi instituída, com o objetivo de emitir, revogar e renovar todos os certificados usados pelos servidores, aplicações, serviços e cidadãos que compõem esta prova de conceito, a "ICP Robson Martins". Essa ICP é formada por uma AC Raiz (a "Autoridade Certificadora Raiz Robson Martins") e uma AC Intermediária ("AC Robson Martins"). Esta última tem, por sua vez, a responsabilidade de emitir os certificados para operação e teste dos sistemas e serviços.

Essa hierarquia de dois níveis foi criada de forma a se assemelhar com uma cadeia da ICP-Brasil, onde a AC Raiz não emite diretamente os certificados, mas sim delega essa atribuição a algumas ACs Intermediárias.

ICP Admin X ICP-Brasil:

A aplicação ICP Admin permite a administração de mais de uma ICP, e pode conter diferentes níveis e combinações de hierarquia.

Como a implementação do ICP Admin segue os padrões e formatos da ICP-Brasil, há uma grande possibilidade (mas não testada) de que se possa utilizar somente certificados digitais emitidos pela cadeia da ICP-Brasil na demonstração desta prova de conceito, de forma a substituir completamente todos os certificados baseados na ICP própria.

APÊNDICE B – IMPLANTAÇÃO NA NUVEM

Um dos requisitos estabelecidos neste trabalho foi a implantação dos sistemas e serviços que compõem a prova de conceito em uma infraestrutura virtualizada na nuvem, de forma a experimentar um serviço de *cloud computing*. Para tanto, foram estudadas algumas opções gratuitas de serviços, chegando-se a duas alternativas possíveis:

- **Google App Engine (GAE)**: Serviço completamente gratuito, que oferece um contêiner de aplicações (que podem ser implantadas até um número máximo simultâneo de dez). Possui ótima integração com os serviços do fornecedor (através das contas do Google), suporte a Java, REST e JPA. Entretanto, não suporta *Web Services* baseados em SOAP e nem permite configurações avançadas no servidor de aplicação (como implementação de JAAS, por exemplo). Também não suporta execução de utilitários de linha de comando, além de outras restrições. Maiores informações estão no endereço: <https://developers.google.com/appengine/> (acesso em 12 nov. 2012).

- **Amazon Elastic Compute Cloud (Amazon EC2)**: Serviço de computação em nuvem que oferece uma infraestrutura de máquinas virtuais completas (com processador, memória, armazenamento e rede), totalmente escalável e redimensionável, e seus custos são cobrados conforme a utilização dos recursos. Permite a instalação rápida e configuração de qualquer sistema operacional (e aplicativos), cujas imagens se encontram disponíveis através do console de administração. Possui uma conta gratuita para experimentação por um ano, o que dá direito (sem custos) a setecentos e cinquenta horas mensais de microinstâncias T1 (suas especificações serão detalhadas mais adiante). Maiores informações podem ser encontradas no endereço: <http://aws.amazon.com/pt/ec2/> (acesso em 12 nov. 2012).

Sendo assim, após a análise de especificações dos dois serviços mencionados, chegou-se a conclusão de que as restrições presentes no serviço *Google App Engine* impediam o atendimento dos requisitos deste trabalho (tais como: impossibilidade de configuração de módulo de login JAAS, de certificados digitais em *keystores* e do uso de certificados próprios para o HTTPS, ausência de permissão para execução do *OpenSSL*, falta de suporte a *Web Services* SOAP, dentre outras).

Desta forma, optou-se pela utilização do serviço *Amazon EC2*, na modalidade gratuita por um ano, pois oferece toda a flexibilidade necessária para a configuração e implantação dos sistemas que compõem a prova de conceito.

A seguir serão detalhados os procedimentos realizados, problemas encontrados e soluções aplicadas durante a experimentação do serviço de computação em nuvem *Amazon EC2*.

Especificações de *hardware*:

Uma microinstância T1 do serviço Amazon EC2 tem a seguinte configuração de *hardware*:

- Até 2 unidades de processamento EC2 (valor máximo de pico, em curtos intervalos de demanda);
- Memória (RAM) de 613 MB;
- Armazenamento somente através de *Elastic Block Store* (EBS);
- Plataforma de 32 ou 64 bits;
- Baixo desempenho de Entrada/Saída (I/O);
- Endereço IP Público dinâmico.

Algumas limitações importantes podem ser deduzidas a partir dessas especificações:

- O desempenho da máquina virtual é bom somente para intervalos muito curtos de picos de demanda. Quando o processador (CPU) da máquina é demandado por um período mais longo de tempo, o desempenho geral cai drasticamente.

- A quantidade de memória RAM é muito baixa se comparada com as máquinas atuais (que geralmente têm mais de 1GB). Isso significa que há menos memória física disponível para o sistema operacional e para as aplicações, o que certamente limita a quantidade possível de programas (processos de sistema) sendo executados simultaneamente.

- O armazenamento é limitado em espaço e em desempenho, ou seja, processos que realizam muitas operações de I/O em disco são executados mais lentamente, e além disso, o espaço total disponível para armazenamento deve ser racionalizado para não se esgotar.

- O endereço IP Público é dinâmico, ou seja, é diferente a cada reinicialização. Há um serviço pago, dentro da Amazon, que permite reservar um IP fixo a uma instância virtualizada, mas não optou-se pela utilização dele neste trabalho por questões de custo.

Essas limitações foram impactantes durante a configuração e implantação dos sistemas que compõem a prova de conceito, pois o servidor JBoss utiliza uma quantidade de memória bastante elevada (em comparação com o total disponibilizado pela microinstância T1) e um nível de processamento muito alto durante o processo de *start-up* e durante a realização do *hot-deploy* de uma aplicação.

O resultado prático foi a percepção de alguns sintomas negativos, como parada brusca e inesperada da execução do servidor JBoss numa frequência muito constante, lentidão excessiva durante o *start-up* e *shutdown* do servidor, e durante o *deploy* de uma aplicação, além de um tempo de resposta muito elevado para responder às requisições de um cliente *web*.

Esses fatos levaram a realização de um estudo, com o objetivo de descobrir formas de otimizar o sistema operacional e o servidor de aplicação para essa configuração limitada de *hardware*.

Após a realização das pesquisas, chegou-se a uma série de procedimentos de configuração, que verdadeiramente levaram à otimização da infraestrutura, e possibilitou o uso dessa conta gratuita do serviço *Amazon EC2*, apesar das limitações impostas por suas máquinas virtualizadas. A seguir estão detalhados esses procedimentos.

Configuração de espaço de *swap* no sistema operacional:

Como a quantidade de memória RAM é pequena, um processo que ultrapasse o espaço disponível certamente será abruptamente encerrado. Para evitar esse problema, existe a possibilidade de se configurar um espaço de troca (*swap*) no sistema operacional, fazendo com que a quantidade de memória seja estendida através de um espaço em disco. O benefício do uso de *swap* é o aumento da memória total disponível para a execução de processos simultâneos, e a grande desvantagem é a enorme lentidão de um disco em comparação com uma memória RAM física.

Por padrão, as imagens de sistema usadas nas microinstâncias T1 da *Amazon* não têm *swap* configurado, pois não é possível particionar o espaço de armazenamento oferecido

Sendo assim, optou-se por configurar o espaço de *swap* como um arquivo dentro da mesma partição onde está o sistema operacional da máquina virtual, conforme o procedimento:

```
$ sudo dd if=/dev/zero of=/mnt/1024.swap bs=1024 count=1048576
$ sudo chmod 600 /mnt/1024.swap
$ sudo mkswap /mnt/1024.swap
$ sudo swapon /mnt/1024.swap
$ sudo echo "/mnt/1024.swap  swap  swap  defaults  0  0" >> /etc/fstab
```

Configuração dos parâmetros de JVM no JBoss:

Devido à pequena quantidade de memória RAM disponível para execução do servidor JBoss, é necessário configurar alguns parâmetros da JVM, de forma a adequar a execução do servidor às especificações oferecidas pelo *hardware* virtualizado. As configurações usadas estão descritas a seguir:

<jboss-home>/bin/run.conf

```
#
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
  JAVA_OPTS="-Xms128m -Xmx512m -XX:MaxPermSize=256m
     -Dorg.jboss.resolver.warning=true
     -Dsun.rmi.dgc.client.gcInterval=3600000
     -Dsun.rmi.dgc.server.gcInterval=3600000"
fi
# Garbage Collection configuration to fix PermGen: OutOfMemoryError(s)
JAVA_OPTS="$JAVA_OPTS -XX:+UseConcMarkSweepGC -XX:+CMSPermGenSweepingEnabled
           -XX:+CMSClassUnloadingEnabled"
```

***Script* para inicialização do JBoss como serviço**:

Para facilitar a realização de *start-up* e *shutdown* do JBoss, foi adicionado um *script* de inicialização para o sistema *Ubuntu Server*, de forma a iniciar e parar automaticamente o serviço JBoss junto com o sistema:

<jboss-home>/bin/jboss_init_ubuntu.sh

```
#!/bin/sh
#
# $Id: jboss_init_ubuntu.sh 0 0 robson@robsonmartins.com $
#
# JBoss Control Script
#
# To use this script run it as root - it will switch to the specified user
#
# Here is a little (and extremely primitive) startup/shutdown script
# for Ubuntu systems. It assumes that JBoss lives in /opt/jboss,
# it's run by user 'jboss'
```

```
# All this can be changed in the script itself.
#
# Either modify this script for your requirements or just ensure that
# the following variables are set correctly before calling the script.

#define where jboss is - this is the directory containing directories log, bin,
#conf etc
JBOSS_HOME=${JBOSS_HOME:-"/opt/jboss"}

#define the user under which jboss will run, or use 'RUNASIS' to run as the
#current user
JBOSS_USER=${JBOSS_USER:-"jboss"}

#make sure java is in your path
JAVAPTH=${JAVAPTH:-"/usr/bin"}

#configuration to use, usually one of 'minimal', 'default', 'all'
JBOSS_CONF=${JBOSS_CONF:-"default"}

#if JBOSS_HOST specified, use -b to bind jboss services to that address
JBOSS_HOST=${JBOSS_HOST:-"0.0.0.0"}
JBOSS_BIND_ADDR=${JBOSS_HOST:+"-b $JBOSS_HOST"}

#define the classpath for the shutdown class
JBOSSCP=${JBOSSCP:-"$JBOSS_HOME/bin/shutdown.jar:$JBOSS_HOME/client/jnet.jar"}

#define the script to use to start jboss
JBOSSSH=${JBOSSSH:-"$JBOSS_HOME/bin/run.sh -c $JBOSS_CONF $JBOSS_BIND_ADDR"}

if [ "$JBOSS_USER" = "RUNASIS" ]; then
  SUBIT=""
else
  SUBIT="su - $JBOSS_USER -c "
fi

if [ -n "$JBOSS_CONSOLE" -a ! -d "$JBOSS_CONSOLE" ]; then
  # ensure the file exists
  touch $JBOSS_CONSOLE
  if [ ! -z "$SUBIT" ]; then
    chown $JBOSS_USER $JBOSS_CONSOLE
  fi
fi

if [ -n "$JBOSS_CONSOLE" -a ! -f "$JBOSS_CONSOLE" ]; then
  echo "WARNING: location for saving console log invalid: $JBOSS_CONSOLE"
  echo "WARNING: ignoring it and using /dev/null"
  JBOSS_CONSOLE="/dev/null"
fi
```

```
#define what will be done with the console log
JBOSS_CONSOLE=${JBOSS_CONSOLE:-"/dev/null"}

JBOSS_CMD_START="cd $JBOSS_HOME/bin; $JBOSSSH"
JBOSS_CMD_STOP=
  ${JBOSS_CMD_STOP:-"java -classpath $JBOSSCP org.jboss.Shutdown --shutdown"}

if [ -z "`echo $PATH | grep $JAVAPTH`" ]; then
  export PATH=$PATH:$JAVAPTH
fi

if [ ! -d "$JBOSS_HOME" ]; then
  echo JBOSS_HOME does not exist as a valid directory : $JBOSS_HOME
  exit 1
fi

echo JBOSS_CMD_START = $JBOSS_CMD_START

case "$1" in
start)
    cd $JBOSS_HOME/bin
    if [ -z "$SUBIT" ]; then
        eval $JBOSS_CMD_START >${JBOSS_CONSOLE} 2>&1 &
    else
        $SUBIT "$JBOSS_CMD_START >${JBOSS_CONSOLE} 2>&1 &"
    fi
    ;;
stop)
    if [ -z "$SUBIT" ]; then
        $JBOSS_CMD_STOP
    else
        $SUBIT "$JBOSS_CMD_STOP"
    fi
    ;;
restart)
    $0 stop
    $0 start
    ;;
*)
    echo "usage: $0 (start|stop|restart|help)"
esac
```

Adição dos certificados digitais da cadeia confiável na JVM:

Para que as aplicações pudessem reconhecer os certificados digitais emitidos pela “ICP Robson Martins”, instituída nesta prova de conceito (especialmente os consumidores de *web services*), foi necessário incluir os certificados da cadeia como confiáveis no *keystore* principal da JVM.

Para realizar isso, foi utilizada a ferramenta *KeyTool*, que acompanha o pacote Java SE, adicionando os certificados da AC Raiz e AC Intermediária (*icprmartins.cer* e *acrmartins.cer*, respectivamente) como autoridades confiáveis dentro do arquivo **<java-home>/lib/security/cacerts**:

```
$ cd <java-home>/lib/security

$ sudo keytool -import -v -trustcacerts -file <path/icprmartins.cer>
  -keystore cacerts -keypass changeit -storepass changeit

$ sudo keytool -import -v -trustcacerts -file <path/acrmartins.cer>
  -keystore cacerts -keypass changeit -storepass changeit
```

Configuração do JBoss para suportar HTTPS:

Abaixo estão descritas as configurações necessárias ao servidor JBoss, para que ele suporte conexões seguras (através de HTTPS), os nomes de domínio e as portas TCP usadas na prova de conceito:

<jboss-server-home>/deploy/jbossweb.sar/server.xml

```
<!-- A HTTP/1.1 Connector on port 8080 -->
<Connector protocol="HTTP/1.1" port="${jboss.web.http.port}"
           address="${jboss.bind.address}"
           redirectPort="${jboss.web.https.port}" />
```

```
<!-- SSL/TLS Connector configuration using the admin devl guide keystore -->
<Connector protocol="HTTP/1.1" SSLEnabled="true"
           port="${jboss.web.https.port}" address="${jboss.bind.address}"
           scheme="https" secure="true" clientAuth="false"
           keystoreFile="${jboss.server.home.dir}/conf/rmartins.jks"
           keystorePass="rmartins" keyPass="rmartins" sslProtocol = "SSL" />
```

<jboss-server-home>/deployers/jbossws.deployer/META-INF/stack-agnostic-jboss-beans.xml

```
<!-- The WSDL, that is a required deployment artifact for an endpoint, has a
       <soap:address> element which points to the location of the endpoint. JBoss
       supports rewriting of that SOAP address.
     If the content of <soap:address> is a valid URL, JBossWS will not rewrite it
       unless 'modifySOAPAddress' is true.
     If the content of <soap:address> is not a valid URL, JBossWS will rewrite it
       using the attribute values given below.
     If 'webServiceHost' is set to 'jbossws.undefined.host', JBossWS uses
       requesters host when rewriting the <soap:address> -->

<!-- Modificado para que o soap:address corresponda ao dominio do servidor. -->
<property name="webServiceHost">tcc.fiap.robsonmartins.com</property>
<property name="modifySOAPAddress">true</property>

<!-- Set these properties to explicitly define the ports that will be used for
       rewriting the SOAP address.
     Otherwise the ports will be identified by querying the list of installed
       connectors.
     If multiple connectors are found the port of the first connector is used. -->
<property name="webServiceSecurePort">443</property>
<property name="webServicePort">80</property>
```

Além disso, o arquivo de *keystore* **<jboss-server-home>/conf/rmartins.jks** foi configurado com o certificado digital e o par de chaves emitidos especialmente para o servidor HTTPS do domínio **tcc.fiap.robsonmartins.com** (é o mesmo certificado configurado no servidor *proxy web* Apache, instalado em outro servidor virtualizado na infraestrutura da *Amazon*).

Uso de um serviço de DNS dinâmico:

Como o endereço IP Público das microinstâncias T1 é dinâmico, ou seja, é diferente a cada reinicialização das máquinas, optou-se por contornar esse problema através do uso de um serviço de DNS dinâmico, chamado de *no-ip*, gratuito para uso pessoal e disponível no endereço: <http://www.no-ip.com/> (acesso em 12 nov. 2012).

Desta forma, o servidor onde o *proxy Apache* está instalado recebeu um cliente *no-ip* para atualização dinâmica e automática de um nome de domínio (esse servidor é o único na arquitetura implementada que tem um ponto de contato com a rede pública da *internet*).

Assim sendo, o servidor DNS do domínio principal `robsonmartins.com` foi configurado para oferecer um subdomínio `tcc.fiap`, apontando para o endereço fornecido pelo *no-ip* ao servidor *proxy Apache* da arquitetura.

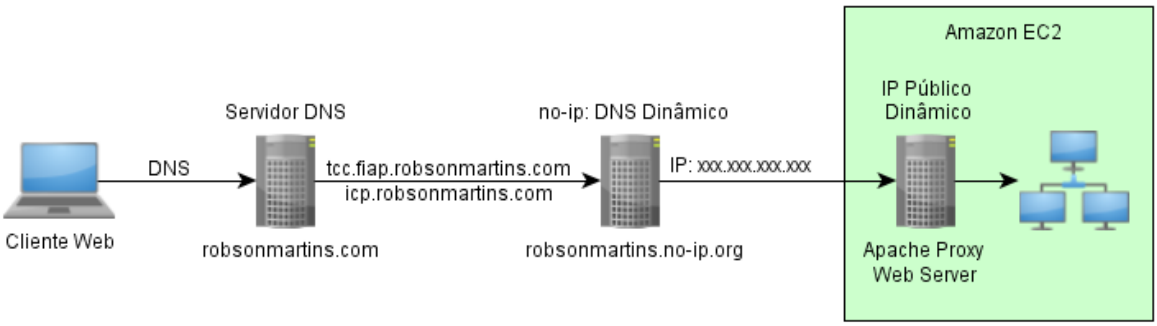

Figura 49 - Arquitetura de servidores DNS

Configuração do servidor *proxy web Apache*:

Para existir um único ponto de entrada de conexões HTTP(S) dentro da arquitetura que forma a prova de conceito, optou-se por configurar um servidor *web Apache* como *proxy*, de forma que todas as requisições pudessem ser repassadas via rede privada (interna) aos servidores JBoss correspondentes a cada aplicação.

Por ser uma configuração mais complexa e fora do escopo do trabalho, este documento não descreve esse procedimento. No entanto, os arquivos de configuração usados no *Apache* acompanham o pacote de códigos-fonte de toda a solução.

Conclusões:

Após a realização das configurações propostas, foi possível uma execução satisfatória do servidor JBoss nas microinstâncias T1, possibilitando o uso do serviço *Amazon EC2* (na modalidade gratuita por um ano) como infraestrutura para este trabalho.

No entanto, devido às limitações já especificadas, há algumas ressalvas:

- O processo de *start-up* do JBoss é bastante lento. Uma reinicialização do servidor pode levar vários minutos até que o serviço esteja no ar.

- O processo de *hot-deploy* de uma aplicação é lento (pode levar até alguns minutos para que a aplicação esteja no ar).

- Os tempos de resposta das requisições de clientes *web* pelo servidor são relativamente altos, mas se mostraram suficientes para a demonstração deste trabalho.

- Por causa das configurações muito limitadas de hardware, os sistemas não estão adequados para oferecer performance e disponibilidade no acesso simultâneo por muitos usuários. Como esse não é o foco desta prova de conceito, essa característica não se torna impeditiva para a demonstração.

- A conta gratuita de experimentação da *Amazon* expira em um ano, a partir da data de sua criação (junho de 2012). Por esse motivo, após esse período, os servidores virtualizados serão desligados e a conta será definitivamente cancelada. Consequentemente, a prova de conceito que demonstra este trabalho não estará mais disponível na *internet* após esse prazo.

APÊNDICE C – PROBLEMAS ENCONTRADOS

Alguns problemas surgiram durante o desenvolvimento da prova de conceito, sendo necessária, então, a aplicação de determinadas soluções de contorno para que eles fossem superados.

Este apêndice relata os problemas mais significativos, e as soluções que foram aplicadas para a sua resolução.

Problemas encontrados durante a implantação na infraestrutura virtualizada na nuvem estão relatados no apêndice B, “Implantação na Nuvem”.

1. Insucesso na implementação de *WS-Security*

Por questões de complexidade de configuração, e incompatibilidade da pilha de *web services* embutida no JBoss 6 (*Apache* CXF) em relação às versões anteriores desse servidor de aplicações, a implementação dos padrões relacionados ao *WS-Security* não foi bem sucedida nesta prova de conceito.

As pesquisas realizadas nesse sentido indicaram que uma das maneiras de se executar WS-Security com sucesso no JBoss 6 é com o uso do *framework Spring*, porém este trabalho não o utiliza em seu desenvolvimento.

Solução aplicada:

Para aplicar segurança aos *web services*, optou-se pelo uso do seguinte:

- Protocolo HTTPS (SSL sobre HTTP), para garantir a criptografia das informações na camada de transporte da rede;
- Autenticação HTTP BASIC, onde o campo *username* contém o conteúdo de um certificado digital (codificado em Base64) e o campo *password* contém a assinatura digital (codificada em Base64) do conteúdo do certificado;
- Proteção do WSDL do *web service* através desse mesmo mecanismo de autenticação.

Como a assinatura digital é realizada com a chave privada do usuário e conferida com a chave pública (que faz parte do certificado), garante-se que o certificado inserido no campo *username* é realmente propriedade do usuário que está tentando efetuar a autenticação (pois só ele conhece a chave privada associada ao seu certificado).

2. Erro na autenticação BASIC em um consumidor de *Web Service*

Quando um consumidor de *web service* tenta se autenticar através do método HTTP BASIC, passando o conteúdo de um certificado digital codificado em Base64 no campo *username*, uma mensagem de erro é obtida, e a conexão é encerrada. A versão do Java utilizada (tanto no servidor onde o *web service* executa, como no consumidor) é a Java 6 (precisamente 1.6.0_37).

Mensagem de erro:

```
...
java.lang.IllegalArgumentException: Illegal character(s) in message header value:
Basic <certificate content in Base64>
at sun.net.www.protocol.http.HttpURLConnection.getInputStream
  (HttpURLConnection.java:930)
at sun.net.www.protocol.http.HttpURLConnection.getHeaderField
  (HttpURLConnection.java:2031)
at java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:376)
at ...
```

Causa:

Após a realização de pesquisas, descobriu-se que a causa desse problema é um *bug* na classe `sun.misc.BASE64Encoder`, que faz parte do Java, conforme relatado no endereço: <http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6947917> (acesso em: 31 jul. 2012).

Este problema então, só acontece nas porções de *software* que executam nos servidores de aplicação da prova de conceito, pois tanto os *web services* como seus consumidores estão implantados lá. O navegador *web* que acessa uma aplicação,

mesmo executando a *Applet* de Autenticação do SICid, estaria imune a esse *bug* (pois a *applet* não contém código que consome *web services*).

Solução aplicada:

Utilização da versão 7 do Java, ao invés da 6, pois ela implementa a correção para esse *bug*, conforme demonstrado no trecho de código-fonte disponível no endereço: <http://hg.openjdk.java.net/jdk7/tl/jdk/rev/a6928350e1f2> (acesso em: 31 jul. 2012).

3. Mensagem de alerta ao iniciar o servidor JBoss 6.1 utilizando o Java 7

Com a utilização do Java 7 ao invés do 6, uma mensagem de alerta (*warning*) passou a ser exibida durante o *start-up* do servidor JBoss 6.1.

Mensagem de alerta:

```
...
WARN [AbstractKernelController] Broken callback: ClassSingleCallbackItem@114fb0a
  {name=interface org.jboss.wsf.spi.metadata.DescriptorProcessor
   whenRequired=ControllerState@90ccda{Installed}
   dependentState=ControllerState@90ccda{Installed}
   attributeName=setProcessor
   owner=AbstractKernelControllerContext@6bedc9
     {metadata=AbstractBeanMetaData@6bf2bf
       {name=WSDescriptorDeployer
        bean=org.jboss.webservices.integration.deployers.WSDescriptorDeployer
        properties=classLoader=BeanMetaDataDeployer$DeploymentClassLoaderMetaData@
          bbb128{classloader=null}
        constructor=null
        autowireCandidate=true
        installCallbacks=
          [method=setProcessor, method=setParser]}
      name=WSDescriptorDeployer
      target=org.jboss.webservices.integration.deployers.WSDescriptorDeployer@
        1f7a8d0 state=Installed depends=AbstractDependencyInfo@10dfdb7{}}
    signature=org.jboss.wsf.spi.metadata.DescriptorProcessor}:
java.lang.ClassCastException:
  org.jboss.wsf.stack.cxf.deployment.jms.JMSDescriptorProcessorImpl cannot be cast
```

```
    to org.jboss.wsf.spi.metadata.webservices.WebservicesDescriptorProcessor
at org.jboss.webservices.integration.deployers.WSDescriptorDeployer.setProcessor
  (WSDescriptorDeployer.java:33) [:6.1.0.Final]
...
```

Causa:

A causa dessa mensagem é uma provável incompatibilidade dos pacotes **`WSDescriptorDeployer`** e **`JMSDescriptorDeployer`** do JBoss 6.1 com o Java 7.

Solução aplicada:

Alteração do arquivo **`<jboss-server-home>/deployers/jbossws.deployer/META-INF/stack-agnostic-jboss-beans.xml`**, de acordo com as instruções sugeridas nesse tópico do fórum de usuários do *JBoss Community*, disponível no endereço: <https://community.jboss.org/thread/195264> (acesso em: 31 jul. 2012).

stack-agnostic-jboss-beans.xml

```
<!-- deployers -->

<!-- Trecho modificado para suportar Java 7:
  <bean name="WSDescriptorDeployer"
        class="org.jboss.webservices.integration.deployers.WSDescriptorDeployer">
    <incallback method="setProcessor"/>
    <incallback method="setParser"/>
  </bean>

  <bean name="JMSDescriptorDeployer"
        class="org.jboss.webservices.integration.deployers.JMSDescriptorDeployer">
    <incallback method="setProcessor"/>
    <incallback method="setParser"/>
  </bean>
-->

  <bean name="WSDescriptorDeployer"
        class="org.jboss.webservices.integration.deployers.WSDescriptorDeployer">
    <property name="processor">
      <inject bean="WSDescriptorProcessor"/>
    </property>
    <incallback method="setParser"/>
  </bean>
```

```
<bean name="JMSDescriptorDeployer"
      class="org.jboss.webservices.integration.deployers.JMSDescriptorDeployer">
  <property name="processor">
    <inject bean="CXFJMSDescriptorProcessor"/>
  </property>
  <incallback method="setParser"/>
</bean>
```

APÊNDICE D – GESTÃO E PLANEJAMENTO DO PROJETO

O desenvolvimento da prova de conceito deste trabalho se constituiu num projeto de pequeno porte, sendo composto de diversos elementos que se caracterizam como um “ciclo de vida de desenvolvimento de aplicações de *software*”.

Como todo projeto de desenvolvimento de *software*, este trabalho foi realizado segundo um planejamento, contendo atividades de gestão, de acompanhamento de cronograma, priorização de tarefas, etc.

Além disso, artefatos gerados ao longo do desenvolvimento do projeto, como códigos-fonte, documentos, arquivos de configuração e outros, foram armazenados e versionados de uma maneira facilmente controlada e recuperável.

Mesmo sendo este um projeto com uma única pessoa alocada na maioria dos papéis (desenvolvedor, analista, líder de projeto, etc.), optou-se por exercitar algumas atividades reais que compõem um processo de desenvolvimento de *software* tradicional.

Sendo assim, para obter o suporte a muitas das atividades que compõem esse processo, foi utilizado um sistema de gestão do ciclo de vida de desenvolvimento (*Application Lifecycle Management* - ALM), mais especificamente o produto *Rational Team Concert* (RTC), fornecido pela IBM / Rational, com licença gratuita para até dez desenvolvedores simultâneos. Abaixo estão descritas algumas características gerais desse produto.

Rational Team Concert (RTC):

O *Rational Team Concert* é uma ferramenta conhecida no mercado por apoiar equipes de desenvolvimento a encarar os desafios diários existentes nos projetos. Possui as principais características:

- Criação e acompanhamento de planos em tempo real para projetos ágeis, tradicionais (formais) ou híbridos;

- Suporte à colaboração entre membros da equipe de desenvolvimento, mesmo que eles estejam em localidades diferentes;

- Gerenciamento de itens de trabalho, com *workflow* (máquina de estados) para tarefas, defeitos, itens de planejamento e outros;

- Versionamento de código-fonte, com possibilidade de automação de *builds* (suporte a integração contínua);

- Suporte a múltiplas plataformas, incluindo *Eclipse*, *Microsoft Visual Studio*, *Microsoft Windows*, *GNU/Linux*, e outros sistemas operacionais e ambientes de desenvolvimento, incluindo *mainframe*;

- Baseado na plataforma *Jazz*, que provê integração com ferramentas de gestão de requisitos, testes, arquitetura, etc.

Maiores informações sobre o RTC podem ser encontradas no endereço <http://www-01.ibm.com/software/rational/products/rtc/> (acesso em 12 nov. 2012).

Projeto TCC FIAP 16SCJ:

Para realizar as atividades de desenvolvimento deste trabalho (tanto as tarefas relacionadas a *software*, bem como outras tarefas), foi criada no RTC uma área de projeto chamada “TCC FIAP 16SCJ”.

Essa área de projeto fornece tudo que é necessário ao planejamento, gestão, desenvolvimento e acompanhamento das atividades relacionadas a este trabalho, incluindo painéis de *dashboard*, editor de planos (cronogramas), visualização de tarefas, *backlog*, áreas de armazenamento e controle de versão (*streams*), etc.

O processo de desenvolvimento utilizado na configuração dessa área de projeto foi o “Processo Tradicional” (Formal), porque o planejamento das atividades foi realizado como um cronograma tradicional, em fases sequenciais.

Abaixo há uma ilustração do *dashboard* principal do projeto “TCC FIAP 16SCJ” no RTC.

Figura 50 - *Dashboard* do projeto no *Rational Team Concert*

Cada elemento presente nesse *dashboard* pode ser configurado de forma a apresentar as informações mais relevantes ao projeto.

Além dos elementos visualizados nesse painel, a ferramenta também oferece a possibilidade de gerar relatórios contendo informações e estatísticas sobre o projeto.

Planejamento do Projeto:

A seguinte linha de tempo foi definida para estabelecer o cronograma macro do projeto “TCC FIAP 16SCJ”:

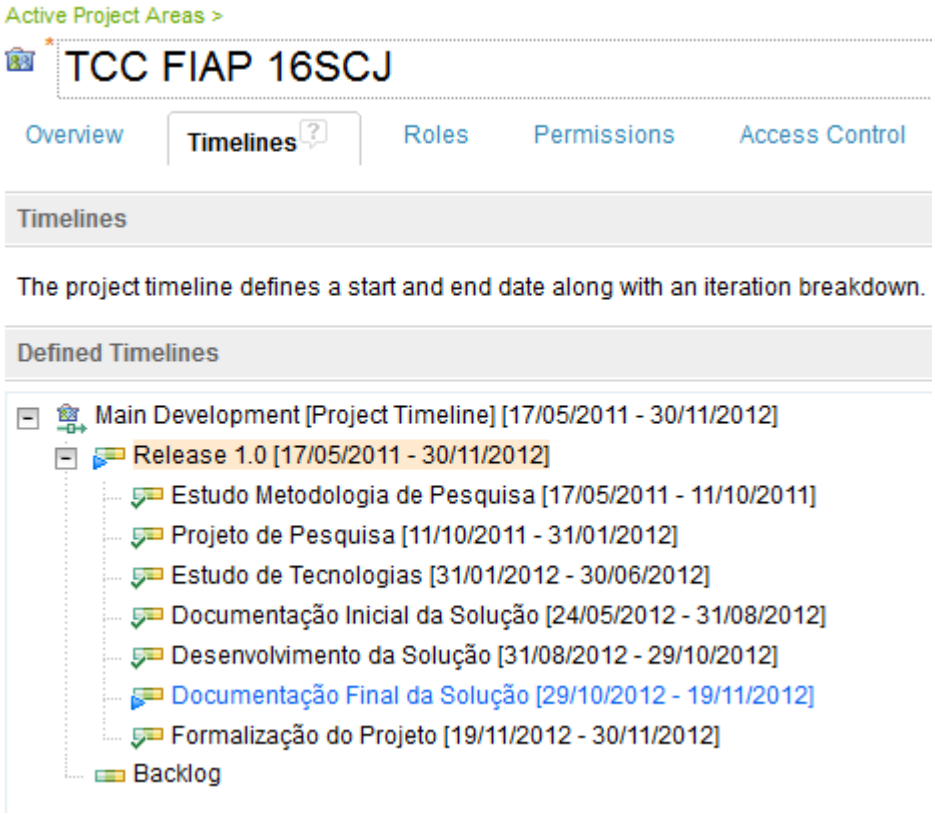

Figura 51 - Linha de tempo principal do projeto no *Rational Team Concert*

Para cada uma das fases definidas na linha de tempo principal, foi estabelecido um plano (cronograma) próprio, contendo todo o conjunto de tarefas (*tasks*), marcos (*milestones)*, e outros itens de trabalho (*work items*).

Como a ferramenta (RTC) permite a alocação de horas por tarefa, é possível acompanhar o andamento do projeto, verificando o tempo gasto em cada atividade, e a diferença para o tempo previsto (planejado). Desta forma, é possível visualizar o impacto no cronograma geral do projeto, e realizar o replanejamento dinâmico das tarefas.

Gerenciamento e controle de versões:

Outro recurso presente no RTC, e amplamente utilizado no desenvolvimento deste trabalho é o sistema de controle de versões.

Para tanto, foi configurado um repositório para controle de versões (*stream*), dentro da área de projeto “TCC FIAP 16SCJ”, e subdividido em alguns componentes (que são unidades lógicas que agrupam artefatos inter-relacionados). Cada um desses componentes contém uma seção de artefatos, tais como: código de sistema, documentação, testes, pesquisa, orientação do TCC, etc.

Para realizar a carga (*load*) e entrega (*deliver*) de artefatos versionados, optou-se por utilizar o ambiente de desenvolvimento *Eclipse*, integrado ao RTC através de um *plugin* próprio a essa finalidade. Assim, todo o desenvolvimento do projeto (incluindo documentação) ficou centralizado numa mesma área de trabalho (*workspace*) do ambiente *Eclipse*, simplificando o acesso aos artefatos.

Outra possibilidade interessante do RTC é a ligação (*link*) entre uma entrega de conjunto de mudanças em artefatos (*change set*) a um item de trabalho (*work item*) correspondente, produzindo um certo grau de rastreabilidade entre os itens (tarefas ou relatos de *bug*, por exemplo) e os artefatos afetados por eles.

Estatísticas de desenvolvimento:

A seguir está uma estatística sobre o desenvolvimento deste trabalho, obtida através de relatórios no RTC. O quadro exibe o tempo dispendido para cada categoria de atividade realizada no projeto “TCC FIAP 16SCJ”, desde sua a concepção, até o momento da elaboração deste documento.

Categoria de Atividade	Tempo Gasto (Horas)
Estudo e Pesquisa	369
Documentação	564
Implementação e Testes (Prova de Conceito)	406
Infraestrutura	42

Quadro 30 - Tempo dispendido por categoria de atividade no projeto