Maurício Simões de Oliveira

Modelagem de um Software Orientado à Componentes para Assinatura Digital

Florianópolis, Santa Catarina

2012.1

Maurício Simões de Oliveira

Modelagem de um Software Orientado à Componentes para Assinatura Digital

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador:
Lucas Ferraro

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis, Santa Catarina

2012.1

Trabalho de conclusão de curso sob o título *"Modelagem de um Software Orientado à Componentes para Assinatura Digital"*, defendido por Maurício Simões de Oliveira e aprovado em 10 de novembro de 2011, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos professores:

Lucas Ferraro
Universidade Federal de Santa Catarina
Orientador

Lucas Ferraro
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Ricardo Felipe Custódio
Universidade Federal de Santa Catarina
Co-orientador

MSc. Cristian Thiago Moecke
Universidade Federal de Santa Catarina
Membro da Banca

MSc. Marcelo Carlomagno Carlos
Universidade Federal de Santa Catarina
Membro da Banca

"Aprender sem pensar é tempo perdido."

Confúcio

Sumário

Lista de Siglas

XMLDSig XML Digital Signature

XML Extensible Markup Language

CMS Cryptographic Message Syntax

ASN.1 Abstract Syntax Notation 1

CAdES CMS Advanced Eletronic Signature

XAdES XMLDSig Advanced Eletronic Signature

PBAD Padrão Brasileiro de Assinatura Digital

ICP Infraestrutura de Chaves Públicas

LCR Lista de Certificados Revogados

AC Autoridade Certificadora

AR Autoridade de Registro

OID *Object Identifier*

SHA-1 *Secure Hash Algorithm 1*

BER *Basic Encoding Rules*

DER *Distinguished Encoding Rules*

MD5 *Message-Digest Algorithm 5*

URI *Universal Resource Identifier*

ACT Autoridade de Carimbo do Tempo

OCSP Online Certificate Status Protocol

LPA Lista de Políticas Aprovadas

DSA *Digital Signature Algorithm*

PA Política de Assinatura

Lista de Figuras

Resumo

Esse trabalho é uma evolução da modelagem apresentado no trabalho Implementação do Padrão Brasileiro de Assinatura Digital (SILVEIRA, 2011). Para evoluir a modelagem foi adotado o paradigma de orientação a componentes. O objetivo dessa mudança é propiciar o reuso dos componentes e expor a modularidade do código. A divisão de componentes adotada tomou como base as especificações dos padrões necessários à implementação do PBAD, dividindo os componentes com base no padrão que cada componente implementa. Como forma de demonstrar os conceitos adotados foi desenvolvido um protótipo. O protótipo ainda serve como um exemplo de referência para futuras implementações. A implementação do protótipo inclui um framework para o suporte da orientação à componentes e a implementação de um software de assinatura digital utilizando esse framework. Concluiu-se com esse trabalho que é viável a implementação de um software de assinatura digital utilizando o paradigma de orientação à componentes.

Abstract

This work is an improvement on the first design proposed for the PBAD (SILVEIRA, 2011). As an improvement on the first design was adopted the component oriented aproach. The main goal is improve the reusability and expose more modules. The division of the components was based on the standards that are utilized by PBAD. To demonstrate the concepts adopt was developed a prototype. The prototype can be a reference for futures implementations. The prototype is composed of a component oriented support framework and a digital signature software that make use of this framework. With is work was concluded that the implementation of a digital signature software utilizing the componente software engineering aproach is possible and can bring benefits.

1 Introdução

A instituição da ICP-Brasil foi feita através da medida provisória 2200-2. O Padrão Brasileiro de Assinatura Digital (PBAD) foi definindo visando normatizar as assinaturas realizadas dentro do ambito da ICP-Brasil. A normatização das assinaturas feita pelo PBAD institui requisitos mínimos que as assinaturas devem cumprir. O PBAD definiu quais os padrões que devem ser utilizados para realizar as assinaturas e as regras sobre as assinaturas. As políticas de assinatura foram adotadas para permitir as aplicações realizar a verificação de conformância das assinaturas na ICP-Brasil. Para simplificar a distribuição das políticas de assinatura, foi desenvolvido a Lista de Políticas Aprovadas (LPA) que não corresponde com um padrão internacional. A LPA é assinada é permite as aplicações localizarem as políticas de assinatura de forma automática.

O PBAD foi desenvolvido com o objetivo de popularizar e possibilitar o uso de assinatura digital no pais. O software de referência do PBAD software foi desenvolvido em duas partes. Uma das partes é a biblioteca para para gerar e validar assinaturas ICP-Brasil, a outra parte é o assinador de referência que implementa um assinador completo utilizando a biblioteca desenvolvida(SILVEIRA, 2011).

Tanto o assinador como a biblioteca cumprem a sua função de propiciar o entendimento do padrão para os desenvolvedores e implementar corretamente o padrão. Porém a implementação impossibilita a separação dos módulos que implementam XAdES dos módulos que implementam CAdES. Como num ambiente de aplicação real o uso dos dois padrões ao mesmo tempo é raro, a utilização da biblioteca do padrão de assinatura digital brasileiro é prejudicada. Outro ponto em que a implementação do padrão peca é a manutenibilidade. Experiências na manutenção do conjunto de artefatos mostraram que o design tanto da biblioteca quanto do assinador estão pouco preparados para absorver as mudanças requiridas ao longo do tempo.

1.1 Objetivo Geral

Propor um novo *design* orientado à componentes para a biblioteca e assinador do PBAD, que esteja mais apto para absorver as mudanças requisitadas ao longo do tempo e que propicie o reuso do padrão.

1.1.1 Objetivos Específicos

- Produzir um framework para suportar a orientação à componentes
- Demonstrar que é possível a implementação de um software de assinatura digital orientado à componentes

1.2 Justificativa

Ao desenvolver um assinador orientado à componentes serão produzidos componentes relacionados com assinatura digital com responsabilidade bem definida e independentes entre si. O problema do acoplamento entre as implementações CAdES e XAdES é resolvido pela separação de um componente para cada. A manutenção também se beneficiará do uso de componentes pois com a alta coesão proporcionada pelos componentes os problemas gerados pelas alterações estarão mais isoladas dentro do software, sendo assim, mais simples de se resolver.

1.3 Metodologia

O desenvolvimento foi baseado em livros e padrões referentes ao assunto tratado. Foi também necessário buscar manuais e outros documentos referentes as ferramentas utilizadas no desenvolvimento. Primeiramente foi feita a pesquisa das informações necessárias para o desenvolvimento do projeto planejado. Então foram selecionadas as ferramentas apropriadas para o desenvolvimento. E então o projeto foi desenvolvido de forma iterativa até que os objetivos propostos fossem alcançados.

1.4 Limitações do Trabalho

Como o PBAD é bastante extenso, bem como os padrões no qual ele é fundamentado, esse trabalho não modela tudo que é necessário para uma implementação completa do PBAD. Em Assinador Orientado à Componentes (capítulo 3), apenas o componente de assinatura CMS Advanced Eletronic Signature (CAdES) foi modelado, deixando do componente de assinatura XMLDSig Advanced Eletronic Signature (XAdES) para um trabalho futuro. Dentre as tecnologias de acesso e gerência de certificados digitais, apenas foi modelado o componente que suporta PKCS#12. Em Protótipo (capítulo 4), alguns componentes foram simplificados. O componente *CMSSignature* 4.2.3 não implementa o suporte aos atributos que caracterizaram o padrão CAdES. A interface com o usuário não possuí o suporte ao manuseio de atributos presentes nos padrões CAdES e XAdES. Portanto o protótipo desenvolvido não é uma assinador que está de acordo com o normativo do PBAD.

1.5 Organização do Trabalho

Esse trabalho se divide em quatro Capítulos, sendo o primeiro a Introdução, o segundo a Fundamentação Teórica, o terceiro Software de Assinatura Digital Orientado à Componentes e o último a Conclusão. A Fundamentação Teórica contém de forma sucinta os conceitos necessários para o entendimento do Software de Assinatura Digital Orientado à Componentes. Em Protótipo é demonstrado a implementação de uma versão simplificada da modelagem proposta. Na Conclusão é descrito aquilo que se percebeu com o desenvolvimento desse trabalho, quais os pontos fortes e fracos do mesmo, e os trabalhos futuros que podem derivar deste.

2 Fundamentação Teórica

2.1 Abstract Syntax Notation 1 (ASN.1)

De acordo com Dubuisson (2000):

> *"Em uma rede com diferentes computadores as mensagens trocadas entre os computadores pode levar à problemas, forçando a implementação de conversores para lidar com problemas como diferentes codificações para textos ou diferentes endianess. Para resolver esse problema de forma mais elegante foi proposta uma notação formal, independente de máquina, que suportasse vários tipos de dados básicos, e construtores de tipos comuns na maioria das linguagens. Por se tratar de uma notação abstrata ela é chamada de Abstract Syntax Notation. Porém, por se tratar de uma linguagem de descrição dos dados, esta não define a semântica dos mesmos, tal semântica fica sob responsabilidade do software que irá processar as estruturas. A idéia é que a notação sirva como forma de comunicação entre os desenvolvedores independente da linguagem que será utilizada ou da arquitetura. A primeira definição desse tipo de linguagem aceita foi chamada de ASN.1."*

Para utilizar as estruturas definidas com ASN.1 é necessário utilizar regras de codificação, que podem ser chamadas de *Transfer Syntax*, pois essa é a representação que será utilizada para realmente para transmitir os dados como stream pela rede. E finalmente a *Concret Syntax* é a representação equivalente dos dados definidos em ASN.1 na linguagem de programação escolhida.

As *Transfer Syntax* são conhecidas como *Encoding Rules*. Dentre as várias *Encoding Rules* temos a *Basic Encoding Rules* (BER), que tem seu valor histórico por ter sido a primeira a ser definida e é amplamente suportada e adotada na implementação de muitos protocolos por codificar os dados de forma independente de arquitetura. Porém, para protocolos de segurança que necessitam assinar dados codificados essas regras de codificação podem causar problemas por possibilitar que um mesmo dado seja codificado de formas diferentes, o que invalidaria uma assinatura por exemplo. Por isso foi definido um subconjunto de regras que tornam a codificação de qualquer dado possível de se representar com ASN.1 único, essa regra de codificação é chamada de *Distinguished Encoding Rules* (DER).

2.2 Criptografia

Nessa seção serão explicados todos os conceitos fundamentais para o entendimento das técnicas de criptografia que serão utilizadas mais adiante nesse trabalho. É explicado o que é criptografia, os aspectos gerais das formas simétricas e assimétricas e finalmente o que é resumo criptográfico.

A palavra criptografia tem em sua raiz o significado de segredo. Criptografia pode ser vista como uma técnica para embaralhar a mensagem, e depois desembaralhar essa mensagem para trazê-la ao seu estado original. Esse ato de embaralhar a mensagem pode visar a proteção da mesma, para mantê-la confidencial e na criptografia moderna pode ser utilizada também para descobrir se ela foi alterada desde a sua criação (HOUSLEY; POLK, 2001).

2.2.1 Simétrica

A criptografia simétrica se baseia em um segredo apenas, conhecido pelos interessados na mensagem. Esse segredo é chamado de chave, e é utilizado em conjunto com algum algoritmo para cifrar uma mensagem qualquer e então depois decifrá-la.

A criptografia simétrica é eficaz para o caso do sigilo. Caso um terceiro intercepte a mensagem, este terá que conhecer a chave, que não é enviada junto com a mensagem, para poder interpretá-la. Se o terceiro desconhece a chave, um bom algoritmo de criptografia simétrica deve ser forte o bastante para que seja inviável computacionalmente especular qual a chave utilizada para cifrar a mensagem (HOUSLEY; POLK, 2001).

Algoritmos de criptografia simétrica podem ser divididos em duas categorias, a cifra de fluxo e a cifra de bloco. As cifras de fluxo operam bit a bit cifrando a mensagem e não necessitam de operações do tipo *padding*[1] , um algoritmo desse tipo é o RC4. Já cifras de bloco operam sobre conjuntos de dados chamados blocos, esses blocos são de tamanhos variáveis e dependem do tipo de algoritmo utilizado. Um algoritmo importante desse tipo é o AES.

2.2.2 Assimétrica

A criptografia assimétrica, que também pode ser chamada de criptografia de chave pública, se caracteriza por possuir duas chaves complementares envolvidas, a chave pública e a chave

[1] *Padding é uma operação realizada antes de cifrar um dado. Considerando o tamanho do dado, a operação de padding consiste em tornar o tamanho do dado um multiplo do tamanho do bloco utilizado pelo algoritmo em questão.*

privada. A criptografia assimétrica geralmente requer um poder computacional maior que a criptografia simétrica para executar a cifra.

Como a criptografia simétrica costuma ser mais simples do ponto de vista computacional, a criptografia assimétrica costuma ser usada para trocar chaves simétricas, para só então criptografar a mensagem desejada. A criptografia assimétrica simplifica muito o processo de manuseio de chaves por possibilitar reduzir o número de chaves que devem ser armazenadas em um ambiente onde, por exemplo, quer se garantir o sigilo da troca de mensagens (HOUSLEY; POLK, 2001).

Um bom exemplo de criptográfia assimétrica é o RSA, que se basea no problema de fatoração de números primos para garantir a segurança de suas chaves. Outro exmplo é o *Digital Signature Algorithm* (DSA), que se basea num problema matemático conhecido como curvas elípticas, o qual também tem elevado grau de dificuldade computacional.

2.2.3 Resumo Criptográfico

Funções de resumo criptográfico mapeiam um conteúdo de tamanho arbitrário para um conteúdo de tamanho fixo. Com a posse do resumo criptográfico é impossível determinar qual a mensagem que originou tal resumo. Porém com a mensagem e o resumo criptográfico sempre é possível afirmar que o resumo criptográfico corresponde a mensagem, por isso o resumo criptográfico costuma ser utilizado para garantir que o conteúdo está intacto. Um exemplo seria uma das técnicas de assinatura digital, que consiste em se obter o resumo criptográfico da mensagem e cifrar este chave privada, para então enviar ao destinatário a mensagem e o resumo criptográfico cifrado. Dessa forma o destinatário pode concluir que a mensagem é aquela que o dono da chave privada assinou (HOUSLEY; POLK, 2001).

Existem varias funções de resumo criptográfico, seu cálculo costuma ser computacionalmente simples. Um algoritmo de resumo criptográfico é o *Secure Hash Algorithm 1* (SHA-1). Outro algoritmo é o *Message-Digest Algorithm 5* (MD5), mas esse teve sua segurança comprometida por tornar-se vulnerável a colisões. (HOUSLEY; POLK, 2001).

2.3 Infraestrutura de Chaves Públicas (ICP)

Uma ICP tem por objetivo distribuir Certificados de chaves públicas, bem como conter informações que possibilitem confirmar a veracidade de tais Certificados. Para distribuir Certificados de forma segura e controlada uma ICP depende de várias entidades, políticas e protocolos de comunicação. Uma ICP pode ser construída de várias formas, uma forma bastante simples é criar um ponto central que contém todas as informações e as gerencia, emitindo Certificados de chaves públicas para os interessados em utilizar a infraestrutura. Essa abordagem não é praticável em ambientes reais, o que nos leva a uma abordagem hierárquica, onde uma entidade delega responsabilidades para outras e se compromete a fiscalizá-las, essas outras entidades se encarregam de partes da informação. A entidade inicial, chamada de raiz, emite um Certificado de chave pública para si própria, e então emite Certificados para as entidades abaixo de si, conferindo a essas entidades algumas permissões. Para conferir a autenticidade da entidade basta verificar o seu Certificado (HOUSLEY; POLK, 2001).

2.3.1 X.509

O X.509 é uma norma publicada pelo ITU-T[2] que define o modelo de ICP. Nesse padrão são definidas as estruturas ASN.1 do Certificado de chave pública e da Lista de Certificados Revogados (LCR). Esse padrão também define quais os protocolos utilizados para comunicação e utilização dos Certificados. Essa norma também define protocolos de manuseio de privilégios, os quais se baseiam em Certificados de Atributo. O objetivo do padrão é prover um esquema de autenticação viável para vários nichos (X.509, 2008).

Certificado de Chave Pública

Um Certificado de Chave Pública tem como principal função associar um nome à uma chave pública. O Certificado também pode conter outras informações como a empresa, ou endereço. Porém associar uma chave pública à um nome não é suficiente. Um Certificado também possui informação de validade, uma vez que as informações contidas em um Certificado digital estão sujeitas a um risco (HOUSLEY; POLK, 2001).

Um Certificado X.509 é emitido por uma AC, por isso é assinado por ela, carregando consigo a informação de qual AC que o emitiu. Portanto um Certificado digital pode ser visto como uma mensagem assinada, com estrutura e dados contidos bem definidos, que servem para

[2]ITU Telecommunication Standardization Sector - Instituto responsável pela padronização de protocolos e outros artefatos para comunicação

distribuir e indentificar sujeitos (HOUSLEY; POLK, 2001).

2.3.2 ICP-Brasil

A ICP-Brasil foi criada com o suporte legal da medida provisória 2.200-2, que confere ao comitê gestor da ICP-Brasil autonomia suficiente para gerenciar a AC-Raiz. Ficou denominado nessa medida provisória que o comitê gestor deveria definir as normas da ICP-Brasil e fiscalizá-las (2.200-2, 2001).

A ICP-Brasil foi desenvolvida utilizando os moldes de certificação digital europeu. Os Certificados e LCRs seguem as recomendações X.509 e as definições da ICP-Brasil visam ser uma restrição sobre o cunjunto de possibilidades previstos no modelo PKIX. Todas as normas necessárias a ICP-Brasil são publicadas na forma de Conjuntos Normativos pelo ITI.

2.3.3 Autoridade Certificadora (AC)

A AC é a entidade mais importante de uma ICP. É ela quem emite os Certificados de chave pública, mantém um registro dos dados e divulga as LCRs. Ela é conhecida pelo seu nome e sua chave pública. A AC também é responsável por manter um histórico dos Certificados já emitidos (HOUSLEY; POLK, 2001).

Quando uma AC emite um Certificado para uma outra AC ou usuário, ela está confirmando que aquele nome indicado no Certificado possui a chave indicada, portanto as ACs devem ter regras restritas quanto à emissão de Certificados, pois essa é uma ação crítica para a confiabilidade da ICP que a AC está inserida. Todos os dados que estão presentes em um Certificado emitido por uma AC serão considerados verdadeiros pelos outros usuários da ICP, por isso o processo de obtenção desses dados deve ser altamente confiável (HOUSLEY; POLK, 2001).

Outro ponto muito importante é a segurança da chave privada da AC. Como esta é uma entidade de suma importância para a ICP, o risco de exposição, ou qualquer outro que a chave privada possa sofrer devem ser minimizados a todo custo. Os dados da AC também devem ser protegidos para que não sejam alterados sem que a ação seja registrada, pois se a AC não possuir seus dados íntegros, a ICP abaixo de si estará comprometida, o que é um enorme dano. Por isso muitas vezes a AC delega o processo de obtenção das informações à uma outra entidade chamada Autoridade de Registro (AR), voltando-se para a segurança da chave privada e dos próprios dados. Isso possibilita uma melhor especialização dentro da AC em assegurar sua própria chave privada e seus dados (HOUSLEY; POLK, 2001).

É definida pela ICP-Brasil dez tipos de Certificados que podem ser emitidos por uma AC,

sendo esses Certificados identificados por um *Object Identifier* (OID) de política de certificação. Dentre os dez tipos de Certificados, quatro se destinam a ACs e usuários finais como pessoas físicas e jurídicas, são nomeados de A1 à A4, diferenciando-se entre si pelos critérios de segurança e sendo esses Certificados com finalidade para uso em assinaturas. Outros quatro tipos de Certificados que se destinam ao sigilo, seguem também a mesma distribuição quanto os critérios de segurança, nomeados de S1 à S4. Por último dois tipos de Certificados destinados a um tipo especial de entidade na ICP, as autoridades de tempo, nomeados T3 e T4 (DOC-ICP-4, 2010).

2.3.4 Lista de Certificados Revogados (LCR)

Os certificados emitidos por uma AC possuem uma válidade pré-definida. Porém podem ocorrer imprevistos antes que a validade do certificado termine tornando o certificado inválido. Através da LCR uma AC pode informar aos usuário da ICP quais dos certificados que ela emitiu já não são mais válidos. Essa LCR é também assinada pela AC que a emitiu e possui uma lista dos certificados que já não são mais considerados válidos e em qual data estes foram revogados. Usualmente as LCRs são emitidas em periodos previsiveis, porém em casos excepcionais uma LCR será emitida antes do previsto. Uma emissão de LCR não deve atrasar pois inviabilizaria a validação dos certificados emitidos pela AC responsável, tornando-os inválidos por falta de informação aos olhos de um verificador.(COOPER et al., 2008)

2.3.5 Online Certificate Status Protocol (OCSP)

O OCSP especifica de forma padronizada uma maneira de verificar a válidade de um certificado junto a um servidor. O servidor OCSP pode ser a AC responsável pelo certificado ou esse serviço pode ser delegado para uma outra entidade. As respostas OCSP fornecidas pelo servidor podem ter sido previamente assinadas ou o servidor pode assinar sempre que a resposta for requisitada (SANTESSON; HALLAM-BAKER,). O OCSP consiste em uma alternativa à LCR (MYERS et al., 1999). O funcionamento do OCSP se diferência do funcionamento da LCR pelo fato de a emissão da LCR ser periódica e a requisição da resposta OCSP caracteziar informação pontual, ou seja, uma resposta OCSP pode ser emitida para qualquer momento no tempo se uma requisição foi feita (MYERS et al., 1999).

2.3.6 Carimbo do Tempo

De acordo com (ADAMS et al., 2001): Um serviço de carimbo do tempo fornece provas de que um dado existiu antes de uma determinada data. Para utilizar carimbos do tempo é

necessária uma terceira parte para fornecer o serviço, a chamada autoridade de carimbo do tempo.

O papel da Autoridade de Carimbo do Tempo (ACT) é fornecer uma evidência de que o dado existiu antes de uma determinada data. Isso é importante dentro da ICP para por exemplo aplicar um carimbo do tempo numa assinatura antes que o certificado do assinante seja revogado, tornando então possível que a chave pública do signatário seja utilizada antes da revogação.

2.3.7 Personal Information Exchange Syntax Standard (PKCS#12)

De acordo com (PKCS. . . , 2000):

> *"O padrão descreve uma Transfer Syntax para informação de indentidade pessoal. O Padrão PKCS#12 suporta varios modos de privacidade e integridade. Os modo de privacidade e integridade mais seguro requer que as plataformas que suportem o padrão possuam pares de chaves confiaveis para a realização de assinaturas digitais. Porém o padrão também possui modos de privacidade e integridade com nível de segurança mais baixa onde o par de chaves pode ser protegido somente por uma senha. O padrão é aplicável tanto para implementações de software como de hardware. Implementações em hardware oferecem segurança física em dispositivos resistentes a violação."*

2.4 Assinatura Digital

A Criptografia Assimétrica provê a base para assinatura digital. Como a chave pública é capaz de decifrar a mensagem cifrada pela chave privada, é possível afirmar quem é o autor da cifra. A assinatura digital consiste em se utilizando da Criptografia Assimétrica, fornecer uma prova de não-repúdio para que o verificador possa identificar o autor. Essa prova é obtida através de um resumo criptográfico cifrado com a chave privada do algoritmo de Criptografia Assimétrica utilizado no esquema de assinatura digital. Podemos citar, por exemplo, um esquema que utiliza RSA e SHA-1, obtém-se o Resumo Criptográfico da mensagem, para então cifrá-lo com a chave privada. Mensagem e Resumo Criptográfico cifrado são enviados para o interessado, que então, para verificar a assinatura, refaz o processo de obtenção do Resumo Criptográfico e decifra o Resumo Criptográfico recebido com a chave pública do autor. Se os dois resumos forem idênticos, então a mensagem é íntegra e pode-se afirmar que o autor da assinatura é o dono da chave privada correspondente à chave pública utilizada para decifrar o Resumo Criptográfico (HOUSLEY; POLK, 2001).

2.4.1 Cryptographic Message Syntax (CMS)

O CMS é um padrão desenvolvido para carregar mensagens. As mensagens podem ser encapsuladas dentro da estrutura CMS de três diferentes maneiras principais, como conteúdo cifrado, assinado e autenticado. Quando o conteúdo é embarcado dentro de uma mensagem CMS como conteúdo assinado, são adicionados outros dados para identificar o esquema de assinatura digital utilizado, e também os assinantes.

A estrutura do CMS é definida em ASN.1, e tem como objetivo possibilitar o processamento da mensagem em um único passo. Os atributos assinados ou autenticados na mensagem CMS devem ser codificados em DER, as outras partes da mensagem não sofrem dessa restrição e podem ser codificados em BER (HOUSLEY, 2009).

Signed-data Content Type

O tipo de conteúdo *Signed-data Content Type* definido no CMS consiste de qualquer tipo de conteúdo e um grupo de assinantes. Varios assinantes podem assinar o conteúdo em paralelo bem como apenas um assinar o conteúdo. O processo de assinatura envolve os passos(HOUSLEY, 2009):

1. Obter o resumo criptográfico da mensagem com o algoritmo desejado por cada assinante do conteúdo;
2. Cada resumo é cifrado com a chave privada do assinante correspondente;
3. Os dados do assinante, e também o resumo cifrado, são incluídos numa estrutura chamada SignerInfo;
4. Todos os SignerInfos produzidos são inseridos juntamente com o conteúdo na estrutura SignedData.

CMS Advanced Eletronic Signature (CAdES)

O padrão CAdES define formatos de assinatura digital, que através da inclusão de atributos assinados ou não assinados na assinatura, prove novos conceitos para a mesma. De acordo com (PINKAS N. POPE, 2008) estes conceitos são:

- Diferenciação de papeis;
- Políticas de Assinatura;
- Dados de validação.

Políticas de assinatura são utilizadas para estabelecer qual a consistência técnica que a assinatura em questão deve possuir. A política de assinatura pode ser utilizada pelo verificador para verificar a consistência técnica da assinatura (PINKAS N. POPE, 2008). Na ICP-Brasil as políticas de assinatura devem ser definidas explicitamente através da inclusão de um atributo na assinatura (ITI, 2010a).

2.4.2 XML Digital Signature (XMLDSig)

O padrão XMLDSig foi desenvolvido com vistas para inclusão de assinaturas digitais em documentos Extensible Markup Language (XML) para web (JR, 1999). Os requisitos seguidos pelo padrão sugerem alguns pontos importantes que as assinaturas XMLDSig devem seguir, de acordo com (JR, 1999) esses requisitos são:

- Assinaturas XML devem ter a sintaxe XML para que possam ser incluídas em documentos XML sem afetar sua estrutura, tornando-a irreconhecivel para as aplicações, ou para que sejam usadas como um documento XML em si;

- O padrão deve ser independente de tecnologia de criptografia utilizada;
- O padrão deve fornecer pelo menos um método de canonização[3], para que o suporte a este seja obrigatório;
- A especificação não pode ser feita de tal forma que seja possível que implementadores a implementem de forma que ofereça algum mecanismo falho.

As XML Signatures possibilitam três modelos de assinatura, sendo eles Detached, Enveloped e Enveloping. O objeto de dados assinado nesses modelos é consecutivamente externo, contém a assinatura e é contido pela assinatura. Todos modelos de assinatura utilizam *Universal Resource Identifier* (URI) para identificar o conteúdo assinado (BARTEL et al., 2008).

XMLDSig Advanced Eletronic Signature (XAdES)

Assim como as extensões CAdES definidas para o CMS, o padrão XAdES visa propor uma especificação para que as assinaturas XMLDSig sejam conformantes com os requisitos necessário à aplicações de documentos eletrônicos que em suma, o não repudio é uma propriedade importante. São definidas propriedades para assinatura através da inclusão de atributos. Estes atributos podem ser assinados ou não, sendo que há definição de quais atributos devem ser assinados e quais não. A especificação é uma extensão compatível com o formato XMLDSig, portanto uma assinatura que é classificada como uma assinatura XAdES também é considerada uma assinatura XMLDSig. Todos os papéis aplicaveis as assinaturas CAdES são também aplicaveis as assinaturas XAdES bem como os conceitos introduzidos ao CAdES são também aplicaveis ao XAdES (CRUELLAS et al., 2003).

2.4.3 Padrão Brasileiro de Assinatura Digital (PBAD)

O PBAD visa proporcionar um padrão para o uso de assinaturas digitais no Brasil (ITI, 2010a). Em (ITI, 2010b) são descritos os requisitos do PBAD, esses requisitos são restrições sobre os padrões CAdES e XAdES. O PBAD possue dez perfis de assinatura, cinco para CAdES e cinco para XAdES. Cada perfil define um conjunto de atributos necessários para garantir a válidade de uma assinatura digital durante um determinado período. A identificação do perfil é obrigatório na ICP-Brasil é ele é feito através da identificação da política de assinatura de forma explicita (ITI, 2010c). Em (ITI, 2010c) é introduzido o conceito de LPA.

[3] O XML necessita de metodos de canonização pois um mesmo conteúdo pode ser representado de diversas maneiras, um método de canonização é capaz de representar um conteúdo numa forma canônica após algum processamento

Lista de Políticas Aprovadas (LPA)

Uma LPA é um arquivo que pode ser processado por máquina que serve para identificar para as aplicações quais as políticas de assinatura são válidas dentro da ICP-Brasil. Há duas LPAs ná ICP-Brasil, uma para assinaturas XAdES definida em XML e outra para assinaturas CAdES definida em ASN.1. Uma LPA contém uma lista de elementos que descrevem as Política de Assinatura (PA)s válidas e revogadas no contexto da ICP-Brasil. A descrição de cada PA consiste de uma referência para um arquivo para o processamento de máquina, um arquivo contendo o texto da política para a leitura do usuário e um resumo criptográfico de ambos os arquivos para cada arquivo referênciado para garantir a integridade dos mesmos. Uma LPA é assinada e tem seu ponto de distribuição fixado para simplificar o acesso por aplicações (ITI, 2010c).

Política de Assinatura (PA)

O PBAD adere um padrão internacional. Há duas versões de PAs, uma para CAdES e outra para XAdES. As PAs são capazes de configurar aspectos da assinatura. Esses aspéctos são configurados através de uma estrutura pré-definida pelos padrões internacionais no qual o PBAD se baseia. As PAs podem definir quais os atributos obrigatórios de uma assinatura, como também restringir qual a ICP à qual os certificados presentes na assinatura podem pertencer. Outras configurações possíveis como restrições de algoritmos utilizados e parâmetros de validação também estão presentes dentro da estrutura de uma PA (ITI, 2010c).

2.5 Padrões de Projeto

De acordo com (HELM, 1995): Sistemas orientados à objetos possuem formas estruturais recorrentes. *Design Patterns*(Padrão de Projeto) são a proposta para documentar e representar essas recorrencias de design no projeto de aplicações. O nome de um *Design Pattern* indica onde este pode ser usado e o seu contexto. *Design patterns* podem ser aplicados em qualquer tipo de sistema, desde a interface até a persistência de uma aplicação. Um *Design Pattern* visa resolver um problema universal. Uma grande porção de um sistema é coberta por *Design Patterns*. Sistemas orientados a objetos podem ser decompostos em subsistemas, e estes subsistemas geralmente são formados por um ou mais *Patterns* nas suas classes principais. Os *patterns* relativos aos relacionamentos entre objetos ajudam a definir a arquitetura do sistema.

O nome de um *design pattern* abstrai e identifica um *design* de estrutura recorrente. O *Design Pattern* se aplica a um problema em questão e esse problema descreve as circunstâncias em que o Pattern pode ser aplicado, se este *Pattern* puder ser aplicado à outros problemas semelhantes, então haverá consequencias e trocas por utilizar-se de um *Pattern* em um escopo mais abrangente do que o previsto.

2.6 Orientação à Componentes

Uma das abordagens existentes para mitigar a alta complexidade do software é a baseada em componentes. Um componente é uma unidade de composição com interface definida por contrato para posterior integração em algum software. O componente define seu comportamento através de interfaces fornecidas e requiridas. Componentes são partes de um sistema que podem ser combinadas a fim de construir um sistema. Um componente só pode ser substituido por outro se estes estiverem em conformidade, ou seja, se o conjunto de interfaces requiridas e fornecidas forem os mesmos e o comportamento externo dos componentes seja semelhante. (OMG. . . , 2010)

Componentes caracterizam o reuso de código. O desenvolvedor não precisa entender como o componente funciona, ele apenas precisa entender como o componente se comporta. Ao desenvolver um componente, há uma troca entre funcionalidade e simplicidade, quanto mais funcional é um componente, ou seja, quanto mais configurável e mais adaptavel é o componente, maior a sua complexidade, portanto será mais difícil entende-lo. Componentes simples, embora muito facilmente entendidos, tem uma aplicação limitada, componentes mais complexos podem ser aplicados em uma gama maior de problemas. (JOHNSON, 1997)

2.7 Frameworks

De acordo com (JOHNSON, 1997): As definições de um framework são vagas, mesmo os frameworks sendo utilizados a muito tempo, ainda não se encontrou uma definição clara do que é um framework. Frameworks podem ser vistos como o esqueleto de uma aplicação, definindo uma estrutura de classes abstratas que modela como as intâncias dessas classes interagem ou como um esboço de uma aplicação que pode ser customizada pelo desenvolvedor para se tornar uma aplicação final.

O reuso de software na era orientada a objetos teve como primeiro interesse o uso de componentes, e posteriormente o reuso de design na forma de patterns. Frameworks são uma forma intermediaria de reuso, que possibilita tanto o reuso na forma de componente, pois o framework inclui código que provavelmente o usuário nunca irá ver, e sim apenas utilizar, como reuso na forma de design, pois a composição do framework em classes abstratas compele a aplicação que se utiliza do framework a se utilizar de um design previamente pensado. O reuso no framework acontece através de classes abstratas e da inversão de controle[4].

Frameworks são como componentes, porém extremamente customizáveis e entretanto de difícil apreendizagem. Quando um framework é bem desenvolvido, ele pode ser útil para desenvolver uma grande gama de aplicações e irá reduzir o tempo necessário para o desenvolvimento delas. Frameworks e componentes devem ser vistos como coisas distintas, embora ambos incitem o reuso, o framework está mais para um contexto reusável para componentes. Frameworks também são similares a outras técnicas de reuso de design, porém frameworks são especificados em linguagem de programação, tornando o aprendizado dos programadores mais simples e eficiente. Design patterns também são formas de se reutilizar o design para problemas especificos, eles padronizam a solução e definem um contexto para sua aplicação, porém além disso frameworks são código, e por isso um framework fornece uma forma do desenvolvedor testar se o seu entendimento do framework está correto. Outro fato é que o tempo gasto para implementar o framework será reaproveitado, tornando o desenvolvimento da aplicação mais rápido.

[4] A inversão de controle, também conhecida como princípio de holywood, é caracterizada pela implementação do fluxo principal da aplicação estar implementado no framework, deixando para o usuário apenas a implementação dos detalhes que serão chamados pelo framwork através de classes abstratas

3 *Assinador Digital Orientado à Componentes*

O software de referência para assinatura digital no Brasil foi implementado em dois módulos principais, uma biblioteca de assinatura digital e o outro um assinador minimalista (SILVEIRA, 2011). Embora tenham mais módulos tanto no assinador minimalista, quanto na biblioteca de assinatura digital, esses módulos não estão explicitados no projeto. A manutenção do assinador minimalista e da biblioteca constitui uma tarefa complexa, pois toda vez que mudanças ocorrem na interface da biblioteca, as alterações são propagadas por todo o código do assinador minimalista e via de regra por outras partes da biblioteca de assinatura digital. A ideia de separar todo esse conjunto de códigos em componentes surgiu da observação de que, embora existam depêndencias, há um número muito maior de módulos que podem ser isolados de forma a estabilizar e explicitar as interfaces entre eles e assim produzir um software com menor grau de acoplamento.

Ao invés de separar o software em duas camadas, ele foi divido em um conjunto de componentes. A interdependência entre os componentes fica restrita as interfaces de comunicação. Portanto graças a natureza da orientação à componentes, o acoplamento do software como um todo é reduzido. Como os componentes são bem isolados uns dos outros, é esperado que mudanças fiquem restritas dentro de um componente. A interação entre os componentes reflete os padrões em que os componentes se baseam, a probabilidade de que esta comunicação mude é bem pequena.

3.1 Design Geral

Para separação de um software em componentes é necessário um critério para poder identificar quais seriam os componentes reutilizáveis dentro de uma organização. Como as partes envolvidas em um software de assinatura digital são padronizadas para propiciar a interoperabilidade, o critério adotado visa separar os componentes segundo a implementação de cada

padrão, com isso se espera uma maior possibilidade de reutilização dos componentes.

O design do assinador foi também inspirado no atual projeto do padrão brasileiro de assinatura digital. O objetivo é de tornar mais simples a reutilização do código já existente, afim de minizar o tempo de desenvolvimento. Isso também se justifica pois à partir da comparação com softwares (CORNELIS, 2012)(CRYPTOLOG, 2012) de assinatura digital que implementam os mesmos padrões, observou-se que algumas das soluções adotadas na implementação atual são recorrentes. Essas soluções recorrentes podem indicar que algumas das soluções adotadas no atual projeto são boas soluções para os problemas encontrados.

3.2 Os componentes do Assinador

O assinador foi dividido em sete componentes, sendo que *SignerGui* (3.2.5), *VerifierGui* (3.2.6) e *IdentityManagerGui* (3.2.7) são componentes de interface com o usuário. O componente *IOGui* (3.2.4) oferece o suporte do sistema para salvar streams de dados no disco. Os componentes *CadesSignature* (3.2.1), *IdentityService* (3.2.2) e *TimeStamp* (3.2.3) implementam os padrões necessários a um assinador digital.

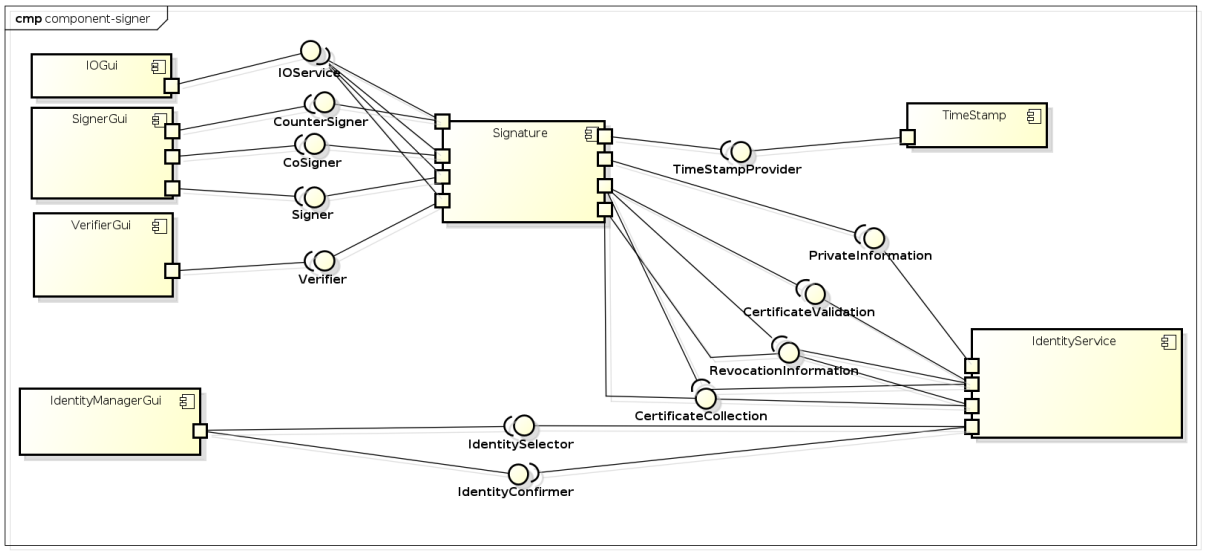

Figura 3.1: Diagrama de componentes do assinador

3.2.1 Signature

O componente *Signature* representa o serviço responsável por realizar e verificar assinaturas. O objetivo de isolar esse componente é permitir a implementação de padrões CAdES e

XAdES de forma que a escolha de qual será utilizado seja configuravel. Nesse trabalho decidimos limitar a modelagem ao padrão CAdES.

CadesSignature

Esse componente tem como responsabilidade implementar o padrão CAdES. O *design* foi baseado na biblioteca de assinatura digital do padrão brasileiro. As funcionalidades de criação dos atributos e instanciação dos geradores, assinaturas e outros detalhes foram incorporados e agora são comandados pelos portos do componente. O componente possui um total de oito portos:

Signer Reponsável pela interação na geração de assinaturas digitais. Implementa a interface *Signer* e depende da interface ;

CoSigner Responsável pela geração de co-assinaturas. Implementa a interface *CoSigner* e depende da interface *IOService*;

CounterSigner Responsável pelo processo de contraassinaturas. Implementa a interface *CounterSigner* e depende da interface *IOService*;

Verifier Responsável pela verificação de assinaturas. Implementa a interface *Verifier* e depende da interface *IOService*;

TimeStamp Porto para comunicações relativas a carimbo do tempo. Depende da interface *TimeStampProvider*;

PrivateInformation Porto para obtenção de depêndencias de informações privadas relativas a identidade e os serviços necessários. Depende da interface *PrivateInformation*;

IdentityInformation Porto para obtenção de informações sobre entidades e validações de entidades. Depende das interfaces *CertificateCollection*, *RevocationInformation* e *CertificateValidation*;

SignatureInformation Porto capaz de exportar os dados que a assinatura pode prover sobre identidades. Implementa as interfaces *CertificateCollection* e *RevocationInformation*.

O porto relativo a geração de assinaturas digitais implementa a interface *Signer*. Essa interface é responsável por possibilitar a um componente qualquer controlar a geração de uma assinatura executando os passos determinados pelo diagrama de estados para esse porto. Esse

porto possui uma dependência, que deve ser fornecida, correspondente aos serviços para realizar o salvamento da assinatura digital. Para realizar uma assinatura o processo efetuado segue os seguintes passos:

1. Definir qual o alvo à ser assinado;
2. Definir a política de assinatura que será utilizada;
3. Concluir a assinatura efetuando a chamada *sign*.

O processo está descrito mais detalhadamente em A.1.

O porto de coassinatura o processo é similar porém o alvo a ser selecionado é a assinatura a ser coassinada. A interface que implementa o processo é a *CoSigner*. Os passos para executar a coassinatura são os seguintes:

1. Definir qual o alvo à ser coassinado;
2. Definir a política de assinatura que será utilizada;
3. Concluir a assinatura efetuando a chamada *cosign*.

O processo está descrito mais detalhadamente em A.2.

O porto de contra-assinatura possui a interface *CounterSignature*. O método para a realização de uma contra-assinatura é também analogo aos citados anteriomente porém para a contra-assinatura é necessário um passo a mais para selecionar a assinatura que será assinada. Os passos são os seguintes:

1. Definir o alvo a ser assinado;
2. Definir qual a assinatura presente no alvo que será assinada;
3. Configurar os atributos;
4. Concluir a assinatura efetuando a chamada *countersign*.

O processo está descrito mais detalhadamente em A.3.

O porto de verificação trabalha de maneira semelhante aos portos citados anteriormente. Os passos para a execução da verificação são os seguintes:

1. Definir o alvo da verificação;
2. Definir qual a assinatura presente no alvo que será verificada;
3. Executar a verificação da assinatura;
4. Consultar os dados da verificação e/ou adicionar atributos na assinatura;
5. Encerrar a verificação através da chamada *clear*.

O processo está descrito mais detalhadamente em A.4.

O porto de comunicação com os serviços de carimbo do tempo realizam chamadas sempre que um atributo do tipo carimbo do tempo é requisitado. O componente de assinatura é responsável por informar qual o resumo criptográfico que deve ser carimbado.

O porto de serviços relativos a identidade tem papel importante tanto nos processos de realização e verificação da assinatura digital. Na realização de uma assinatura, o componente de assinatura irá requisitar ao serviço de manuseio de identidades uma chave privada para concluir a assinatura. O serviço de manuseio de identidades é responsável por fornecer o acesso à essa chave e confirmar a identidade do usuário. O componente de assinatura digital apenas realiza a assinatura com a chave recebida. A gerência de qual chave será utilizada é de responsabilidade do componente de manuseio de identidades e é transparente ao componente de assinatura. Em ambos os momentos o serviço de manuseio de identidades é responsável por fornecer ao componente de assinatura acesso aos certificados de entidades que são requisitadas e também processar validações dos certificados dessas entidades.

O componente possui um porto que fornece acesso aos certificados presentes em uma assinatura após alguma ter sido selecionada para verificação ou execução da contra-assinatura. O processo de verificação de assinaturas sempre utiliza esse serviço, não sendo necessário conectar manualmente a interface CertificateCollection no porto de serviços relativos a identidade.

3.2.2 IdentityService

Componente que representa a abstração de um componente responsável por implementar o acesso as informações sobre entidades, tanto de certificação quanto de revogação. O objetivo de isolar esse componente é permitir que diferentes tecnologias sejam utilizadas para esse fim. Dentre as tecnologias possíveis de se utilizar podemos citar Cryptographic Token Interface

Standard (PKCS#11)[1], PKCS#12, Network Security Services (NSS) [2], Repositório de certificados do *Windows*. Nesse trabalho decidimos limitar o escopo e apenas projetar o componente responsável por prover a funcionalidade necessária através da tecnologia PKCS#12.

PKCS12Repository

Componente que implementa os serviços de acesso à informações de identidade através da tecnologia PKCS#12. A classe *PKCS12PrivateInformationImpl* responsável pelo acesso às informações privadas implementa as interfaces de diferentes portos. Há a dependência com a interface *IdentityConfirmer*, que deve ser fornecida por algum componente externo. Caso essa dependência não seja fornecida o acesso a chave privada, por exemplo, não será liberado e o componente irá negar o acesso às informações privadas. A parte responsável pela realização da validação do caminho de certificação poderá ser reutilizada por outras implementações do serviço de identidade e pode ser vista como um componente interno.

3.2.3 TimeStamp

O serviço de requisição de carimbo do tempo para uma ACT é um padrão bem estabelecido ref (ADAMS et al., 2001). Esse componente é responsável pela gerência das configurações de conexão e negociações via rede com a ACT necessárias para a obtenção de carimbos do tempo.

Esse componente não possui estados de operação devido a natureza do serviço que ele fornece. A requisição do carimbo do tempo é atômica e assim que requisitado o componente pode devolver o carimbo de tempo ou avisar que não obteve sucesso.

3.2.4 IOGui

Componente que fornece ao componente de assinatura o suporte para salvar assinaturas. Ele é responsável por interagir com o usuário do assinador e salvar a assinatura se requisitado no lugar que o usuário indicou. Esse serviço está presente pois para uma maior versatilidade do componente de assinatura. A realização da interface *IOService* é responsável por iniciar as interfaces com o usuário e então realizar as decisões tomadas por ele. O serviço que esse componente fornece é implementado com apenas uma chamada na interface *IOService*.

[1] O PKCS#11 define uma interface padrão para acesso dispositivos criptograficos.

[2] O NSS é um conjunto de bibliotecas desenvolvidas para suportar aplicações de segurança multiplataforma (MOZILLA, 2012).

3.2.5 SignerGui

O componente SignerGui é responsável pela interação do usuário com o assinador. Não é obrigatório que o componente de assinatura forneça todas as três interfaces *Signer*, *CoSigner* e *CounterSigner*. Isso é justificado pelo fato de que nem todas as tecnologias podem oferecer suporte a todas as interfaces, como por exemplo o XAdES, em que a Co-Assinatura só pode ser realizada de modo implícito em condições especiais[3].

A interface com o usuário se adapta as conexões feitas com o componente de assinatura e mostra ao usuário apenas as ações possíveis. A classe principal do componente, a classe *SignerGui*, é responsável por controlar as transições de estado e executar as chamadas nas interfaces do componente de assinatura na ordem correta.

3.2.6 VerifierGui

A interface com o usuário para executar a verificação de assinaturas é realizada por este componente. De forma análoga ao componente anterior, a classe principal do componente, a classe *VerifierGui*, é responsável por gerenciar as transições de estado e chamar na ordem correta as operações na interface *Verifier*. Como este componente depende apenas da interface *Verifier*, cumprir essa dependência é obrigatório.

3.2.7 IdentityManagerGui

Nesse componente estão duas interfaces com o usuário, a *IdentityManagerWindow* e *IdentityManagerConfirmerWindow*, necessárias para a utilização do componente *IdentityManager*. A funcionalidade realizada pelas classes *IdentityManagerGui* e *IdentityManagerWindow* a interação com o usuário para possibilitar a seleção de uma identidade digital. A classe *IdentityManagerGui* é responsável por executar as chamadas na interface *IdentitySelector* na ordem apropriada e a classe *IdentityManagerWindow* é responsável por exibir as informações necessárias ao usuário.

A classe *IdentityManagerConfirmerWindow* implementa a interface *IdentityConfirmer*, está provê um serviço necessário ao componente *IdentityManager* que confirma a identidade do usuário através da solicitação de alguma senha que liberá o acesso à chave no *IdentityManager*, independente de qual a tecnologia utilizada.

[3]Devido a estrutura da assinatura XMLDSig, uma co-assinatura só é possível em um contexto em que o modo de assinatura é Enveloped e uma assinatura não irá interferir na outra.

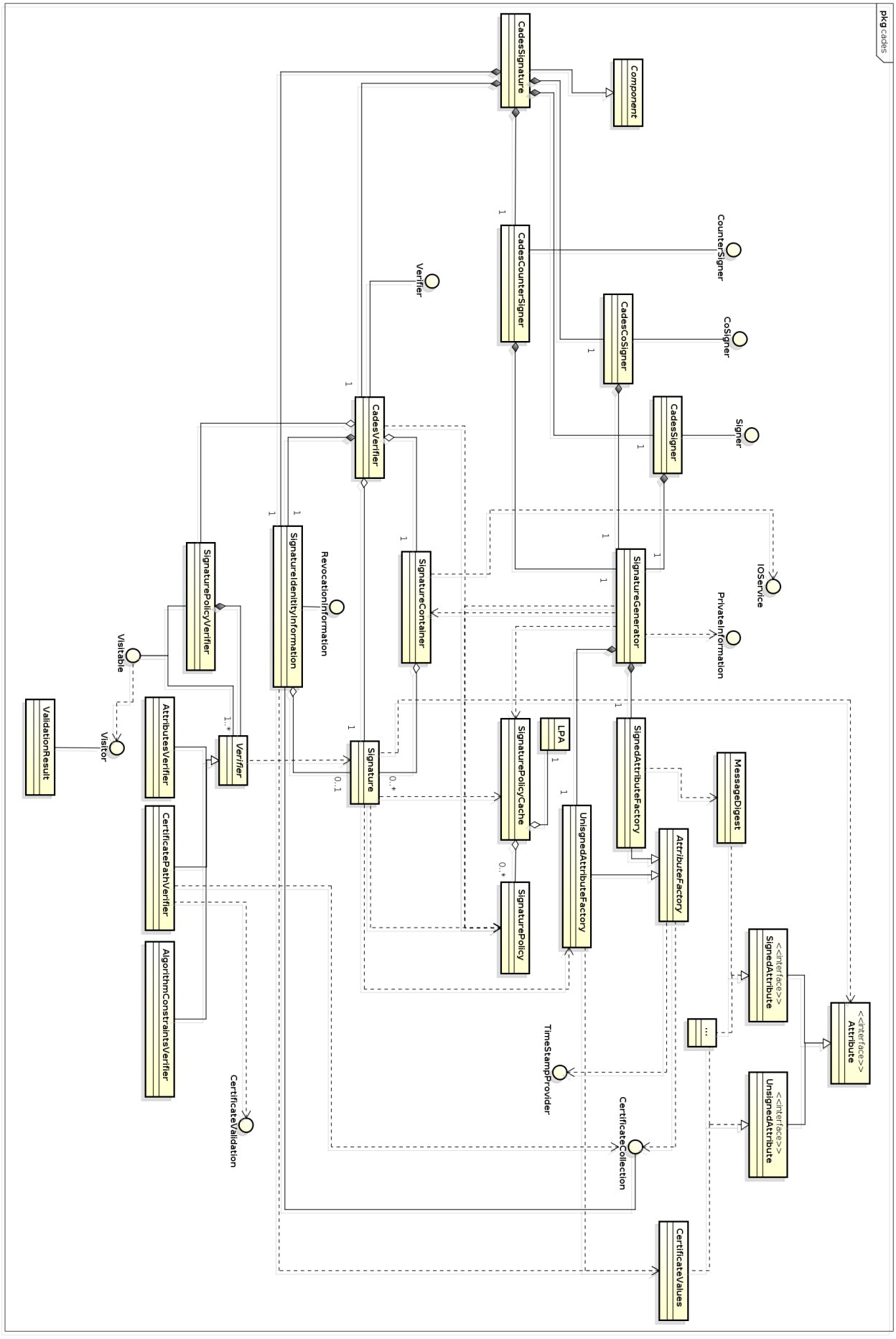

Figura 3.2: Diagrama de classes do componente *CadesSignature*.

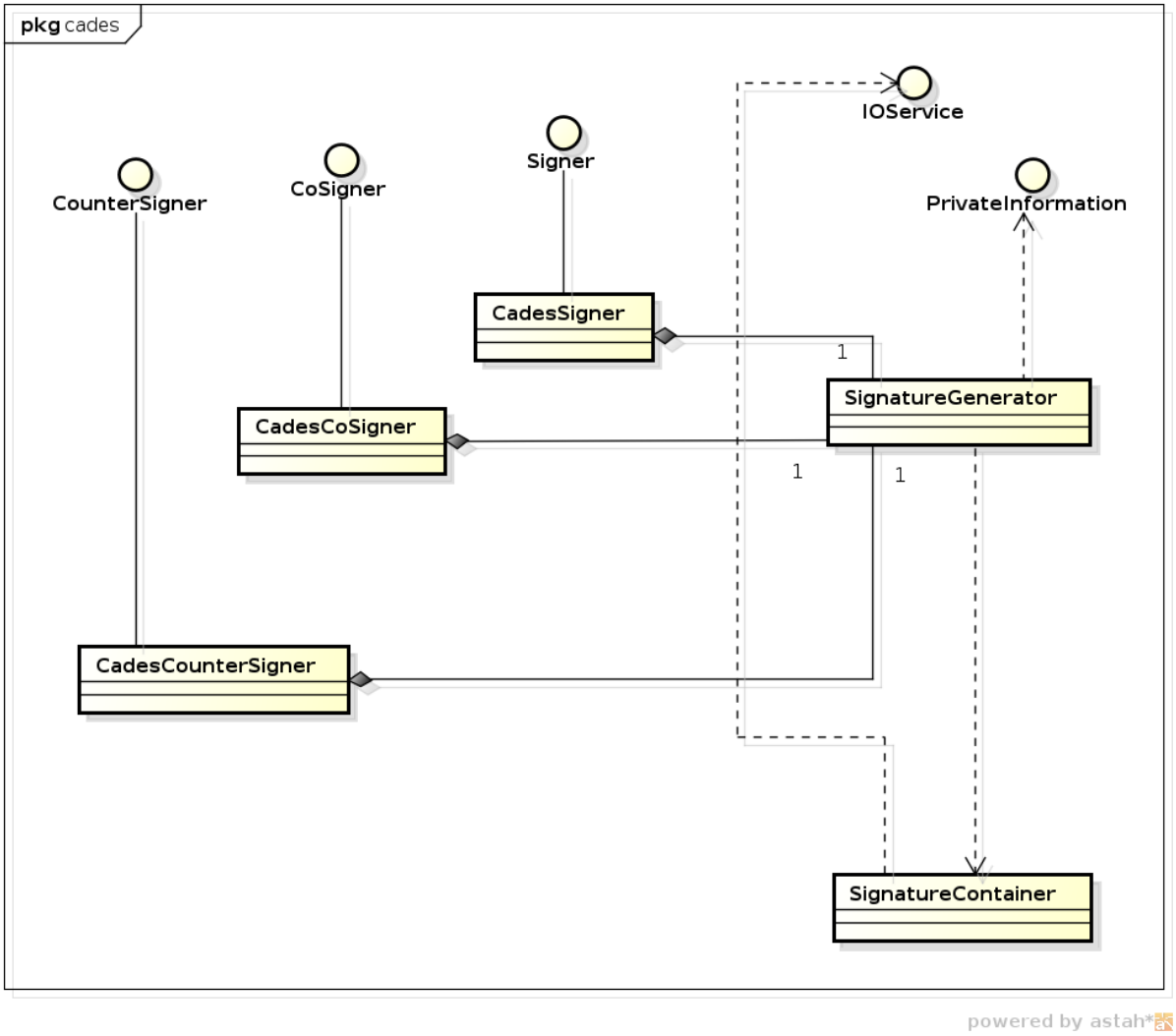

Figura 3.3: Diagrama das classes que implementam os serviços de assinar, co-assinar e contra-assinar.

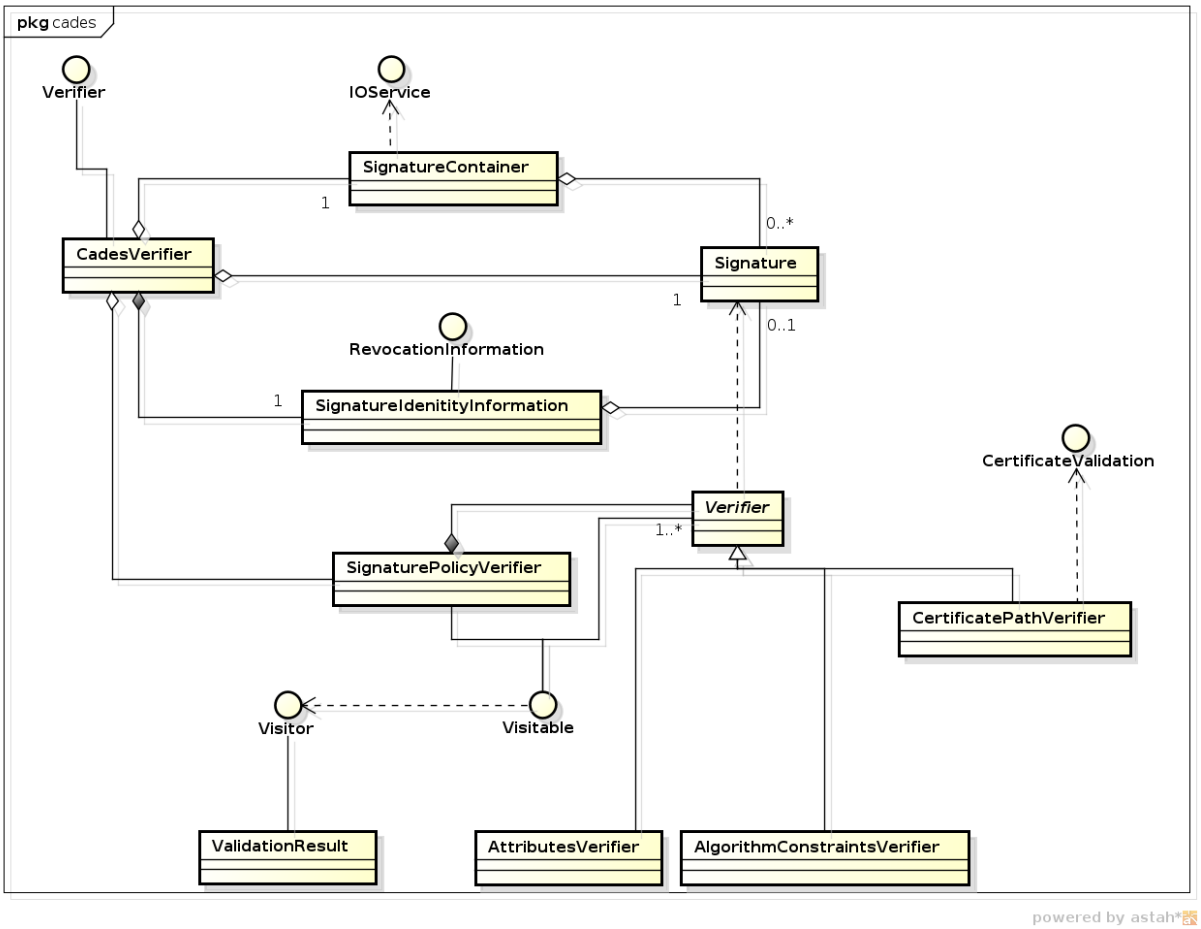

Figura 3.4: Diagrama das classes que implementam o serviço de verificação de assinatura.

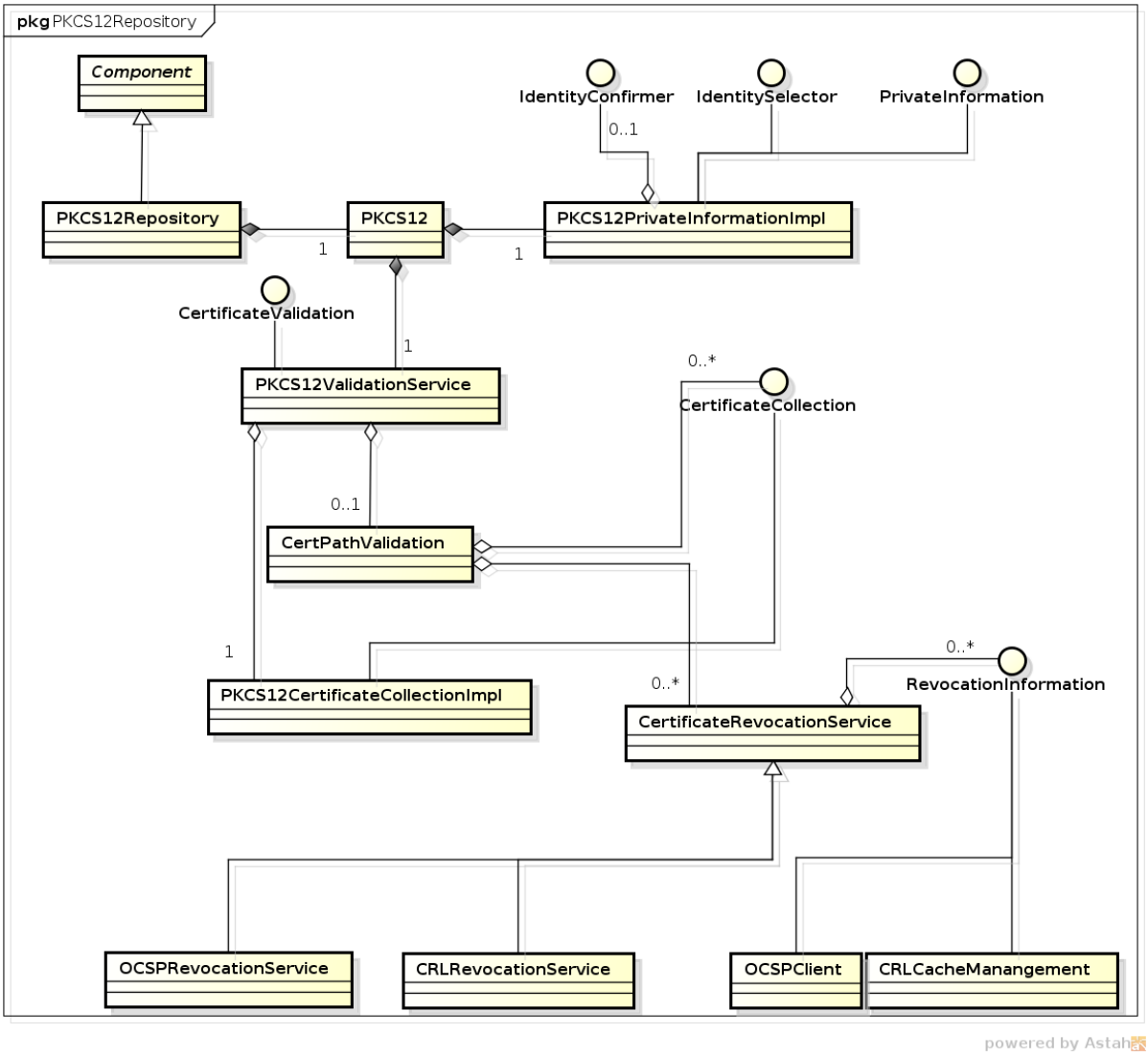

Figura 3.5: Diagrama de classes do componente *PKCS12Repository*.

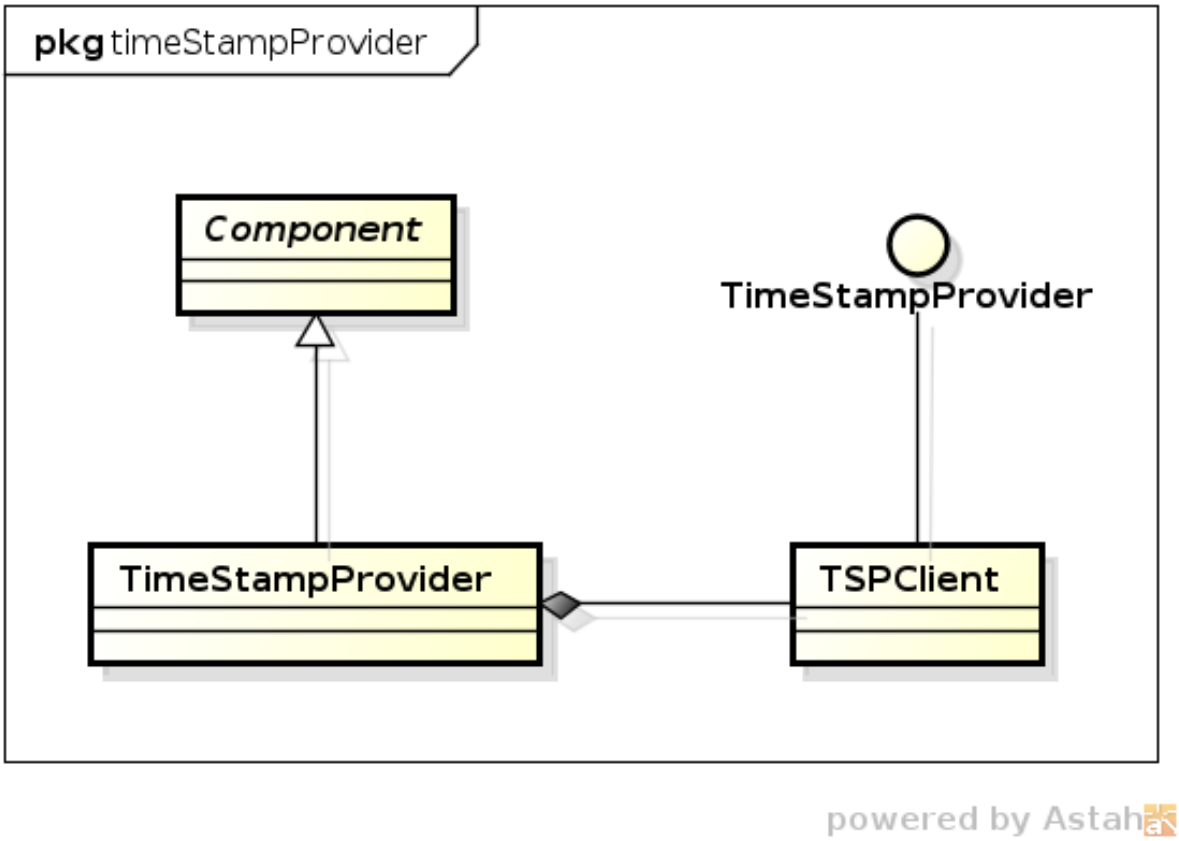

Figura 3.6: Diagrama de classes do componente *TimeStampProvider*.

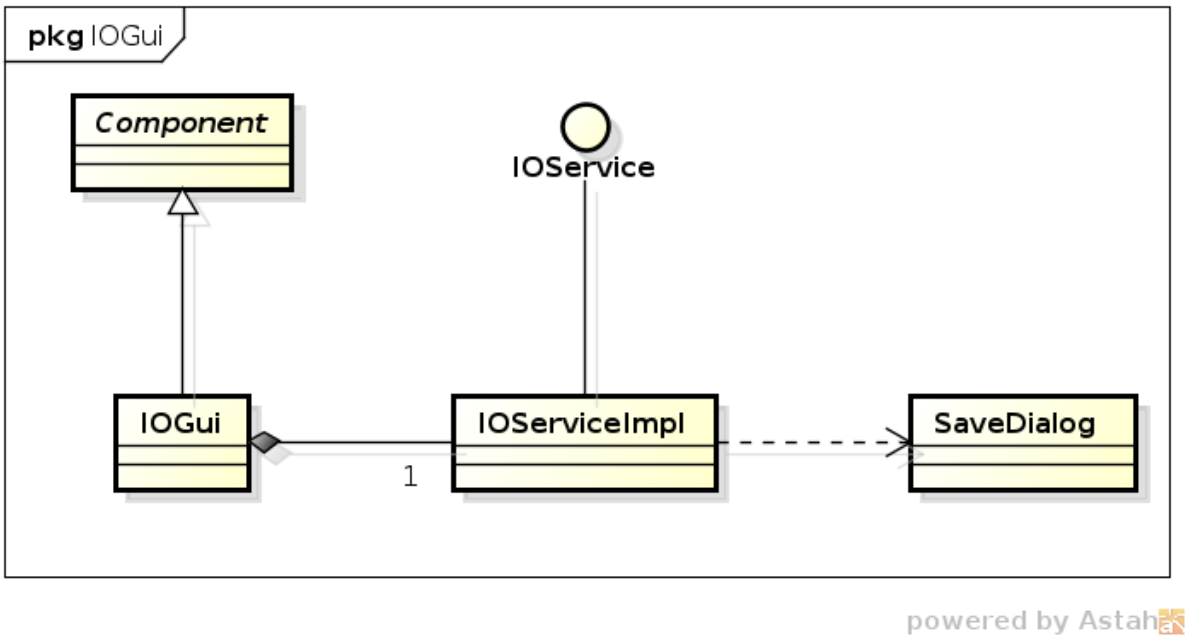

Figura 3.7: Diagrama de classes do componente *IOService*.

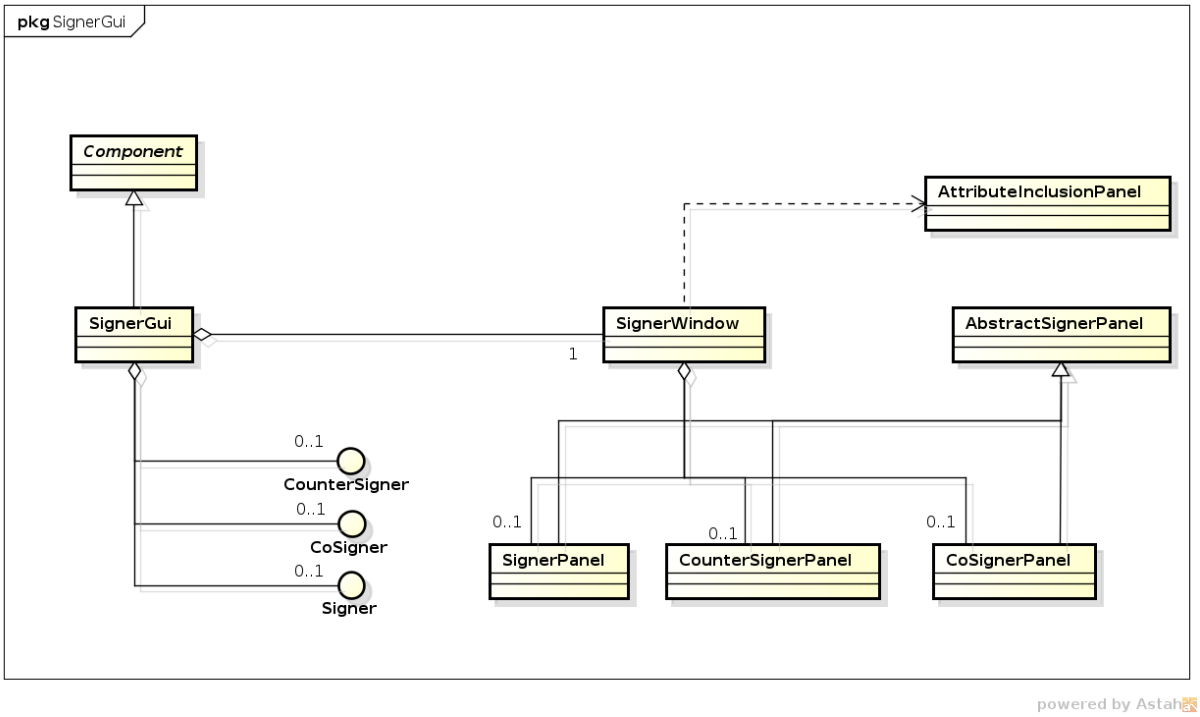

Figura 3.8: Diagrama de classes do componente *SignerGui*.

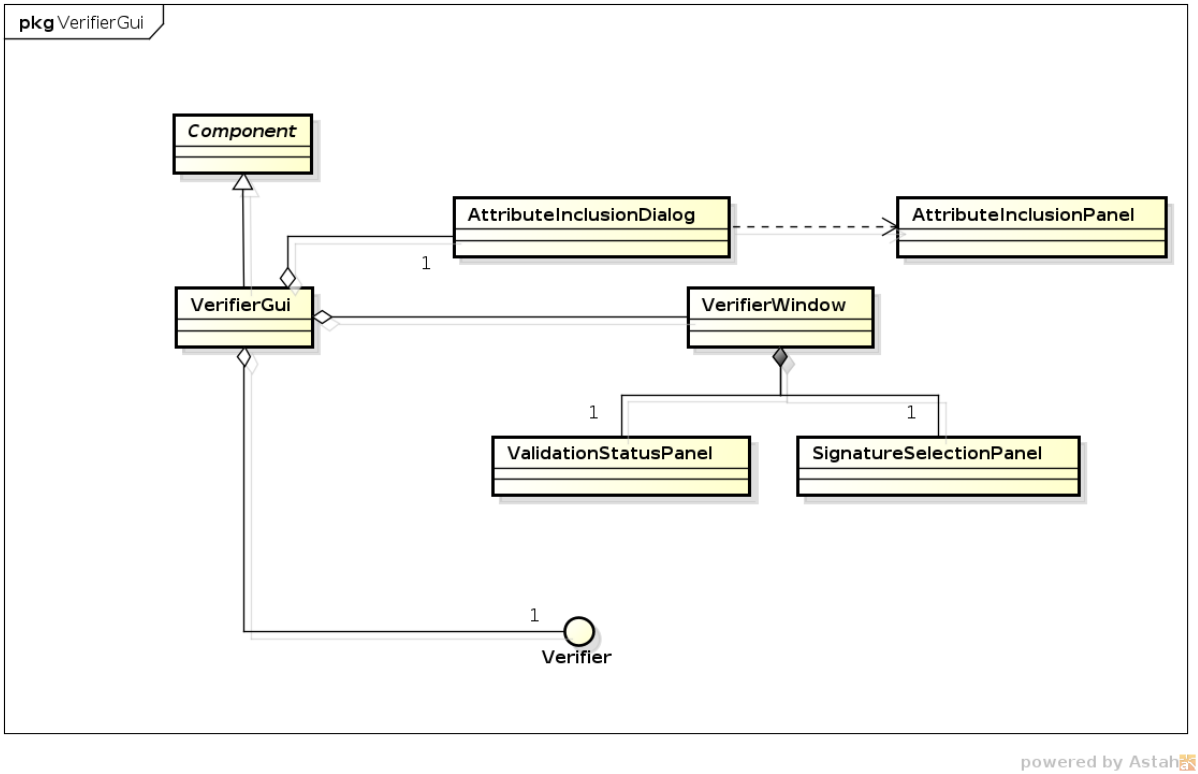

Figura 3.9: Diagrama de classes do componente *VerifierGui*.

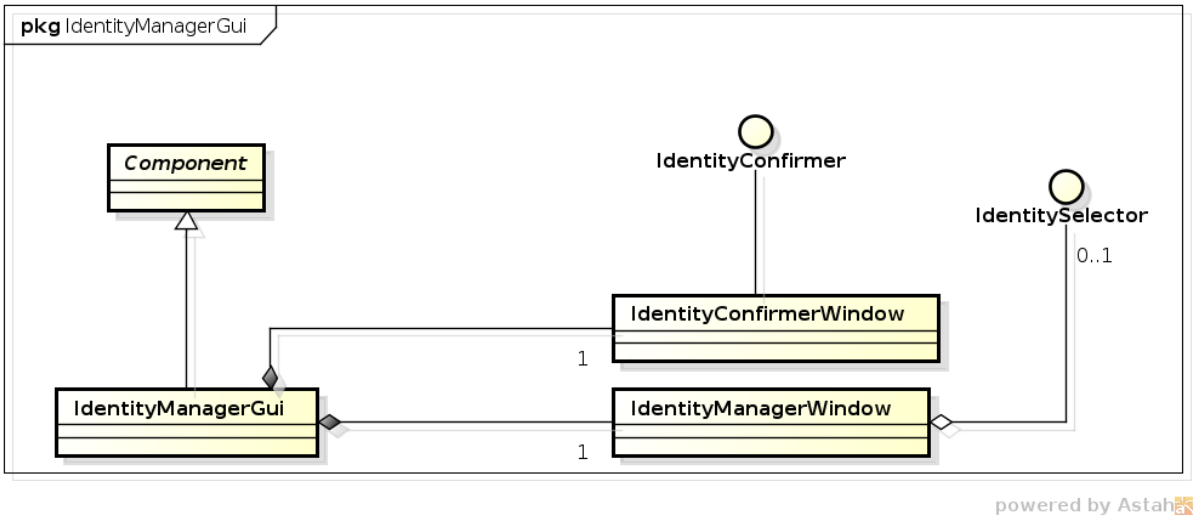

Figura 3.10: Diagrama de classes do componente *IdentityManagerGui*

4 Protótipo

O protótipo visa provar a viabilidade do desenvolvimento de um software orientado à componentes para assinatura digital. Para tal foi desenvolvido um *framework* que suporta a orientação à componentes e um assinador simplificado capaz de realizar assinaturas CMS. O *design* do Mini Assinador é uma simplificação do proposto no capítulo Assinador Orientado à Componentes(3).

4.1 Framework de suporte

O *framework* de suporte implementa as noções fundamentais da orientação à componentes. Para tornar possível a composição do software através de componentes o *framework* fornece uma classe padrão. Essa classe representa um componente que deve ser estendida para criar um componente. A configuração das ligações e dos componentes de um software são feitas através de um arquivo de configuração, uma descrição mais detalhada e um exemplo do arquivo está no Anexo B. O *framework* é capaz de a partir do arquivo de configuração instanciar os componentes e conecta-los.

Esse *framework* é extremamente simples, e sua implementação foi feita através de três classes e uma interface que implementa o conceito de *Annotation***??**. As classes do framework são descritas a seguir:

Component É uma classe abstrata e implementa alguns métodos fundamentais que informam se a configuração do componente é suficiente para o seu funcionamento.

Application Implementa a inicialização da aplicação e é responsável por executar a configuração. Nela está a implementação da instanciação dos componentes e a conexão dos portos.

ComponentsConfiguration Implementa a interpretação do arquivo de configuração e expõe essa configuração de uma forma simples para a classe *Application*

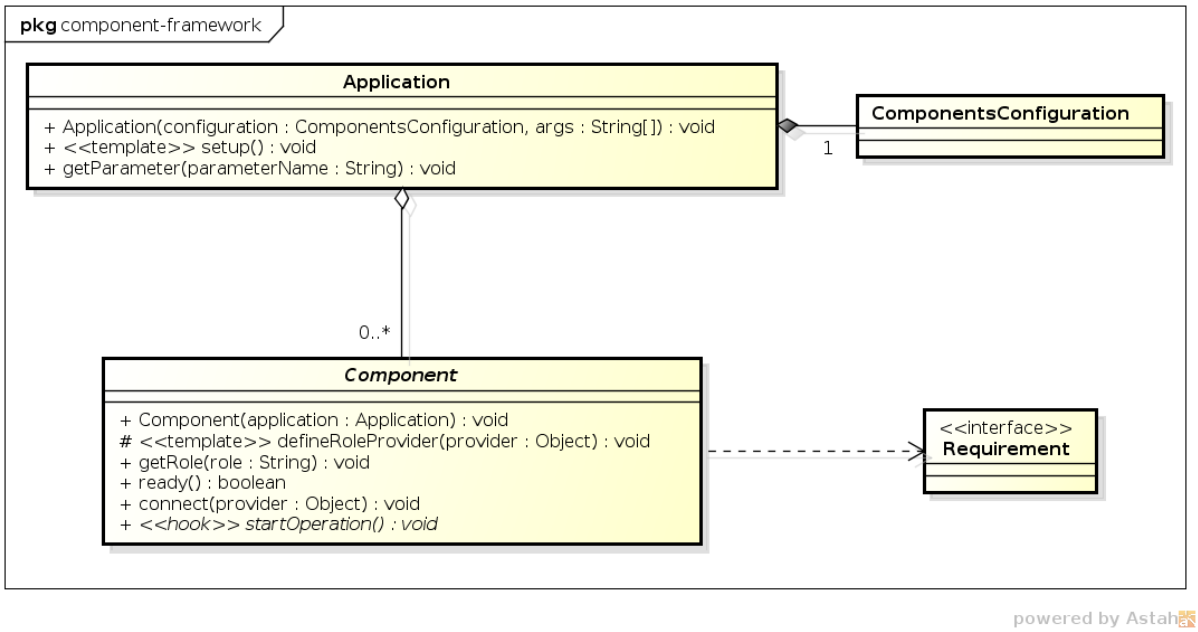

Figura 4.1: Diagrama das classes do framework de suporte.

Requirement É a anotação utilizada para especificar nos componentes onde a conexão dos portos deve ser efetuada. Ela é utilizada pela classe *Component* para efetuar as conexões dos portos.

O *Framework* não implementa as verificações de compatibilidade funcional entre os componentes. Os problemas de incompatibilidade funcional devem ser identificados e resolvidos manualmente.

4.2 Mini Assinador

O assinador desenvolvido é constituído pelos seguintes componentes:

- SignerGui
- VerifierGui
- CMSSignature
- IOGui
- IdentityManagerGui
- PKCS12Repository

Esses componentes foram selecionados para implementar uma versão simplificada do assinador, o componente CMSSignature é uma simplificação do componente CadesSignature pois não implementa a extensão CAdES para a assinatura CMS. Foi feita a integração do Mini Assinador com o navegador de arquivos *Nautilus*.[1]

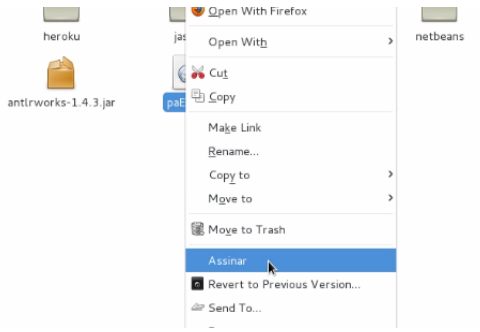

Figura 4.2: Menu para realizar a assinatura.

4.2.1 SignerGui

O componente SignerGui implementa apenas o suporte para assinaturas. Embora o padrão CMS suporte também co-assinaturas, visando simplificar o protótipo, o suporte a co-assinaturas não foi desenvolvido. A única conxão que o componente SignerGui aceita é a interface *Signer*.

O funcionamento do componente *SignerGui* do protótipo é análogo ao do componente SignerGui(3.2.5), com exceção do suporte a configuração de atributos, que não foi implementada no protótipo.

4.2.2 VerifierGui

Esse componente implementa a interface para a visualização da verificação de integridade das assinaturas CMS e da validade do certificado que realizou a assinatura. A única conexão aceita por esse componente é a interface *Verifier*. O seu funcionamento é análogo ao do componente VerifierGui(3.2.6), porém muito mais limitado, uma vez que não há o suporte para a adição de atributos não assinados, nem o suporte as outras diversas verificações que se fazem necessárias para se adequar ao PBAD.

[1] *Nautilus* é um navegador de arquivos das distribuições linux.

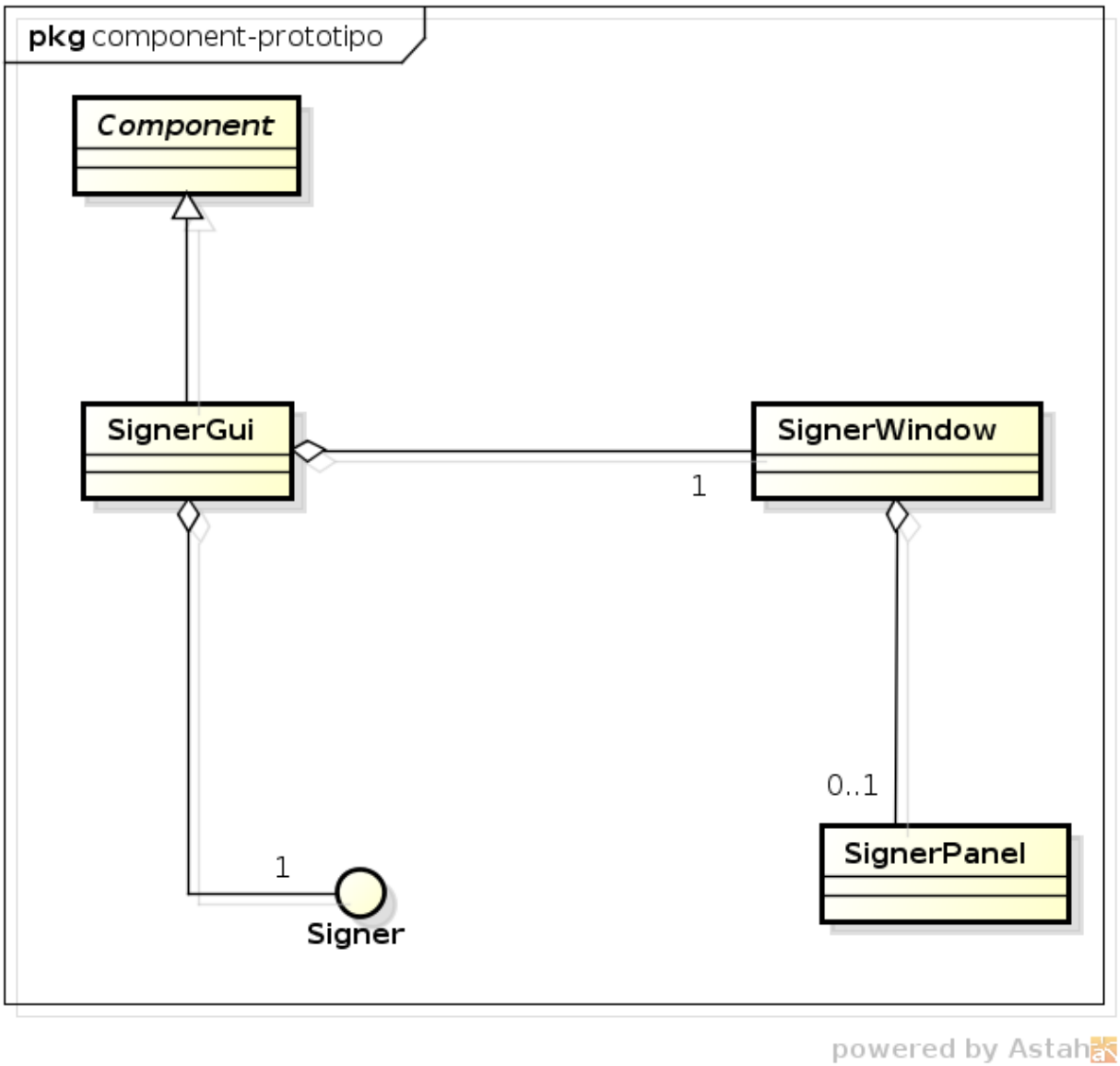

Figura 4.3: Diagrama de classes do componente *SignerGui* do protótipo.

4.2.3 CMSSignature

A implementação do padrão CMS é feita por esse componente. Ele fornece as interfaces *Signer* e *Verifier* é depende das interfaces *PrivateInformation*, *CertificateValidation* e *IOService*. A estrutura de classes é análoga à do componente *CadesSignature*(3.2.1), porém bastante simplificada.

4.2.4 IOGui

O componente *IOGui* é o mesmo que componente que já foi descrito em 3.2.4. Por se tratar de um componente bastante simples, nenhuma alteração visando a sua simplificação foi

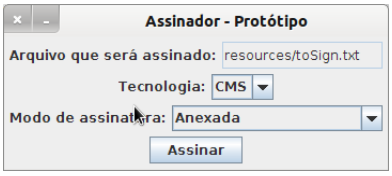

Figura 4.4: Interface de usuário implementada pelo componente *SignerGui*.

necessária.

4.2.5 IdentityManagerGui

Esse componente também não foi simplificado e é o mesmo que já foi descrito em 3.2.7. Não foi possível uma simplificação desse componente pois o serviço que ele implementa é essencial mesmo para a versão simplificada desse assinador. As interfaces com o usuário que esse componente implementa são duas:

- Janela de seleção de Identidade Digital
- Janela de confirmação de Identidade

4.2.6 PKCS12Repository

A implementação desse componente também não foi simplificada e é a mesma que a descrita em 3.2.2. As funcionalidades fornecidas por esse componente são essenciais para o bom funcionamento do mini assinador.

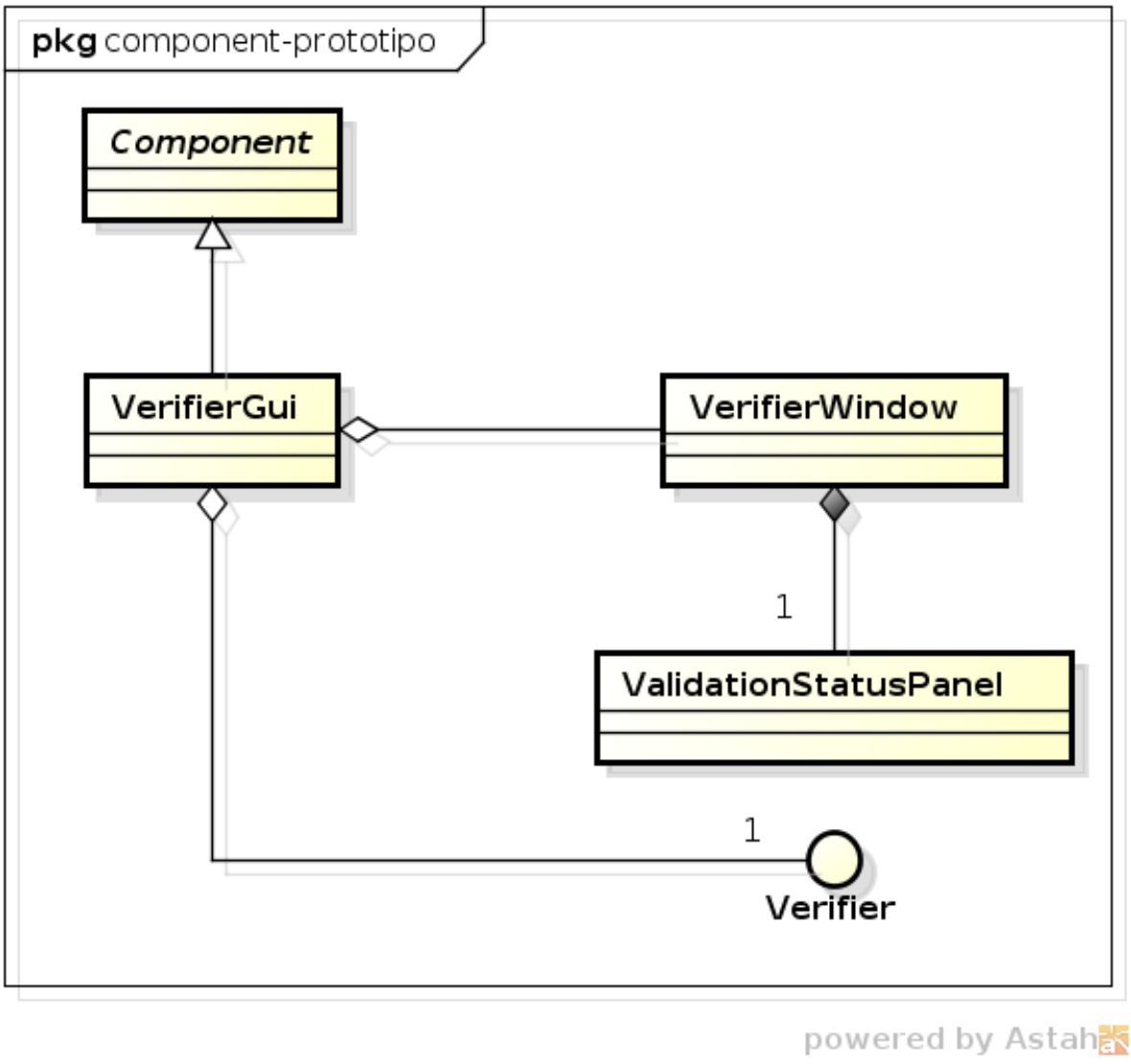

Figura 4.5: Diagrama de classes do componente *VerifierGui* do protótipo.

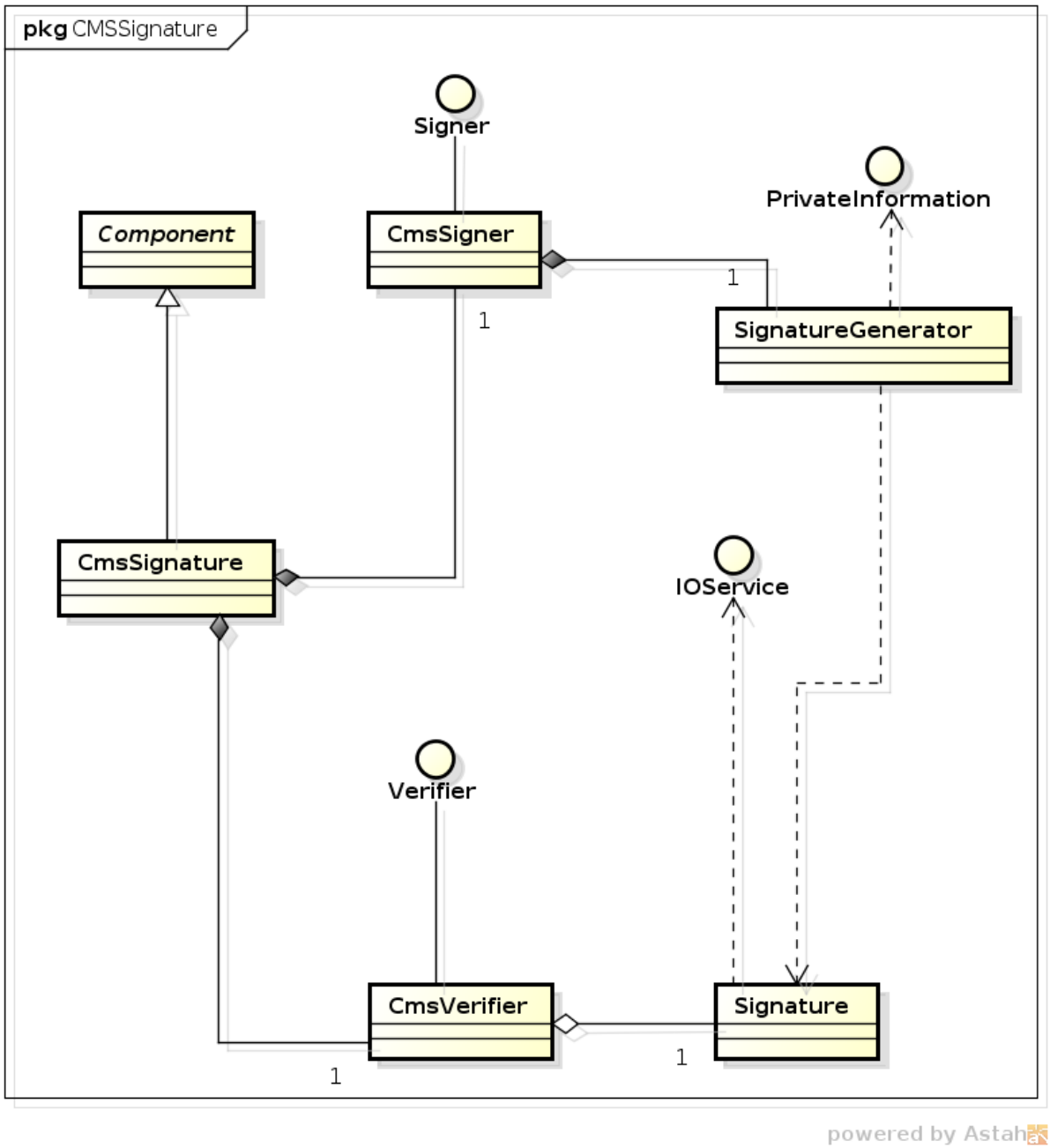

Figura 4.6: Diagrama de classes do componente *CMSSignature* do protótipo.

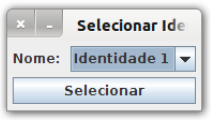

Figura 4.7: Janela de selação de identidade.

5 *Conclusão*

A modelagem proposta no capítulo Assinador Orientado à Componentes (3) é modular devido a natureza do paradigma de orientação à componentes. Pelo fato da alta modularidade do software o esperado é que as mudanças que ocorrerem sejam mais isoladas e impliquem em menos problemas no software como um todo. Componentes com uma interface bem definida e documentada são amigaveis ao desenvolvedor. Como o comportamento do componente é bem definido é provavel que algum desenvolvedor se sinta mais avontade e entenda mais facilmente essa interface sem se preocupar com os detalhes de implementação.

O protótipo desenvolvido (4) demonstra que é possível escrever um software de assinatura digital seguindo o paradigma de orientação à componentes. O modo de interação com o usuário foi ligeiramente modificado, ao invés do usuário abrir o assinador para então selecionar o arquivo ele seleciona o arquivo no seu navegador de arquivos e então executa o assinador sobre aquele arquivo.

Para desenvolver o protótipo foi necessário o desenvolvimento de um pequeno framework. Embora o framework propicie uma orientação à componentes de forma mais explícita ele pode limitar as possibilidades de reuso do componente, uma vez que para utilizar os componentes é necessário utilizar também o framework, ou então, reimplementar tudo aquilo que o framework faz.

Uma das principais dificuldades encontradas durante o desenvolvimento do trabalho foi a modelagem de interação entre os componentes. Não há uma maneira de se estabelecer a forma de interação entre os componentes que será a melhor sem ser a experiência no domínio.

5.1 Trabalhos Futuros

Nesse trabalho foi modelado um conjunto de componentes necessários para a implementação do padrão CAdES. Como continuação desse trabalho fica a modelagem dos componentes para implementação do padrão XAdES e dos outros padrões de acesso à informação de identi-

dade.

Foi desenvolvido um framework para suportar os componentes no protótipo. A avaliação dos benefícios e deficiências dessa abordagem caracteriza uma continuação desse trabalho.

A modelagem realizada nesse trabalho abrange apenas uma das etapas do desenvolvimento de um software. O processo de desenvolvimento adotado deve levar em consideração o fato do software em desenvolvimento ser orientado à componentes. Um estudo do processo de desenvolvimento orientado à componentes que melhor se adeque aos conceitos abordados aqui é também um ponto em aberto.

Referências Bibliográficas

2.200-2. *MEDIDA PROVISÓRIA Nº 2.200-2, DE 24 DE AGOSTO DE 2001*. 2001. Disponível em: <http://www.iti.gov.br/twiki/bin/view/Certificacao/MedidaProvisoria>. Acesso em: 2012-11-22.

ADAMS, C. et al. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. IETF, ago. 2001. RFC 3161 (Proposed Standard). (Request for Comments, 3161). Updated by RFC 5816. Disponível em: <http://www.ietf.org/rfc/rfc3161.txt>.

BARTEL, M. et al. *XML Signature Syntax and Processing (Second Edition)*. 2008. Disponível em: <http://www.w3.org/TR/xmldsig-core/>. Acesso em: 2012-11-22.

COOPER, D. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. 2008. Disponível em: <http://www.ietf.org/rfc/rfc5280.txt>. Acesso em: 2012-11-22.

CORNELIS, F. *eID Applet*. 2012. Http://code.google.com/p/eid-applet. Disponível em: <http://code.google.com/p/eid-applet>. Acesso em: 2012-11-22.

CRUELLAS, J. C. et al. *XML Advanced Electronic Signatures (XAdES)*. 2003. Disponível em: <http://www.w3.org/TR/XAdES/>. Acesso em: 2012-11-22.

CRYPTOLOG. *CUTE*. 2012. Disponível em: <http://www.cryptolog.com/en/products/toolkit-/cute-electronic-signature>. Acesso em: 2012-11-22.

DOC-ICP-4, I. *REQUISITOS MÍNIMOS PARA AS POLÍTICAS DE CERTIFICADO NA ICP-BRASIL*. 2010. Acesso em: 2012-11-22.

DUBUISSON, O. *ASN.1 Communication Between Heterogeneous Systems*. [S.l.]: Morgan Kaufmann, 2000. ISBN 0126333610.

HELM, R. Patterns in practice. 1995.

HOUSLEY, R. *Cryptographic Message Syntax (CMS)*. 2009. Disponível em: <http://tools-.ietf.org/html/rfc5652>. Acesso em: 2012-11-22.

HOUSLEY, R.; POLK, T. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. 1. ed. [S.l.]: Wiley, 2001. ISBN 0471397024.

INSTITUTO NACIONAL DE TECNOLOGIA DA INFORMAÇÃO. *Visão Geral Sobre Assinaturas Digitais na ICP-Brasil*. v. 2.0. Brasília, Abril 2010. DOC-ICP-15.

ITI. *Requisitos Mínimos para Geração e Verificação de Assinaturas Digitais na ICP-Brasil*. v.2.0. Brasília, Abril 2010. DOC-ICP-15.01.

ITI. *Requisitos Mínimos para Políticas de Assinatura Digital na ICP-Brasil*. v.2.0. Brasília, Abril 2010. DOC-ICP-15.03.

JOHNSON, R. E. Components, frameworks, patterns. 1997.

JR, J. R. (Ed.). *XML-Signature Requirements*. 1999. Disponível em: <http://www.w3.org/TR-/xmldsig-requirements>. Acesso em: 2012-11-22.

MOZILLA. *Network Security Services*. 2012. Http://www.mozilla.org/projects/security/pki/nss/. Acesso em: 2012-11-22.

MYERS, M. et al. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. IETF, jun. 1999. RFC 2560 (Proposed Standard). (Request for Comments, 2560). Disponível em: <http://www.ietf.org/rfc/rfc2560.txt>.

OMG Unified Modeling LanguageTM (OMG UML), Superstructure. 2010. Disponível em: <http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF/>. Acesso em: 2012-11-22.

PINKAS N. POPE, J. R. D. *CMS Advanced Electronic Signatures (CAdES)*. 2008. Disponível em: <http://tools.ietf.org/html/rfc5126>. Acesso em: 2012-11-22.

PKCS 12 v1.0: Personal Information Exchange Syntax. [S.l.], 2000. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2138>. Acesso em: 2012-11-22.

SANTESSON, S.; HALLAM-BAKER, P. *Online Certificate Status Protocol Algorithm Agility*. Http://tools.ietf.org/html/rfc6277. Disponível em: <http://tools.ietf.org/html/rfc6277>. Acesso em: 2012-11-22.

SILVEIRA, L.

Implementação do Padrão Brasileiro de Assinatura Digital — Ciências da Computação,Universidade Federal de Santa Catarina, 2011.

X.509, I.-T. *Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks*. 2008. Disponível em: <http://www.itu.int-/rec/T-REC-X.509-200811-I/en>. Acesso em: 2012-11-22.

ANEXO A – Diagramas de iteração entre interfaces

A.1 Assinar

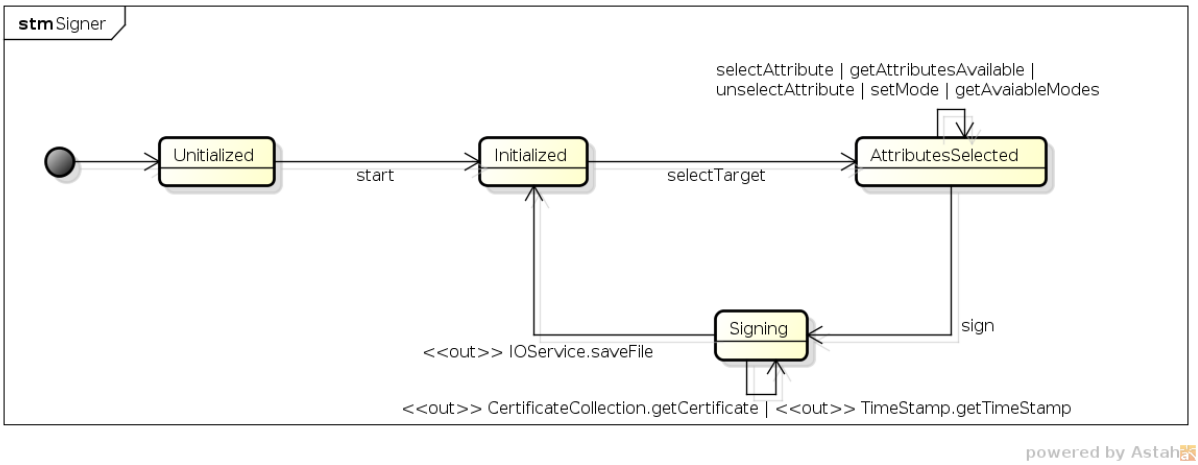

Figura A.1: Diagrama de estados para representar a ordem das chamadas para realizar uma assinatura.

1.Defina qual o alvo à ser assinado;

2.Defina a política de assinatura que será utilizada;

É possível nesse momento configurar qual o modo que deve se executar a assinatura, no caso *Attached* ou *Detached*;

Os atributos disponíveis para inclusão na assinatura podem ser obtidos através da chamada *getAvailableAttributes*;

A inclusão de um atributo na assinatura é comandada pela chamada *selectAttribute*.

3.Conclua a assinatura efetuando a chamada *sign*.

Aqui a configuração da assinatura é verificada;

A assinatura será gerada se a configuração for a correta. As chamadas para obter os dados necessários para a construção de atributos é feita nesse momento;

O componente de assinatura irá chamar o componente *IOGui* através de sua interface para realizar o salvamento da assinatura.

A.2 CoAssinar

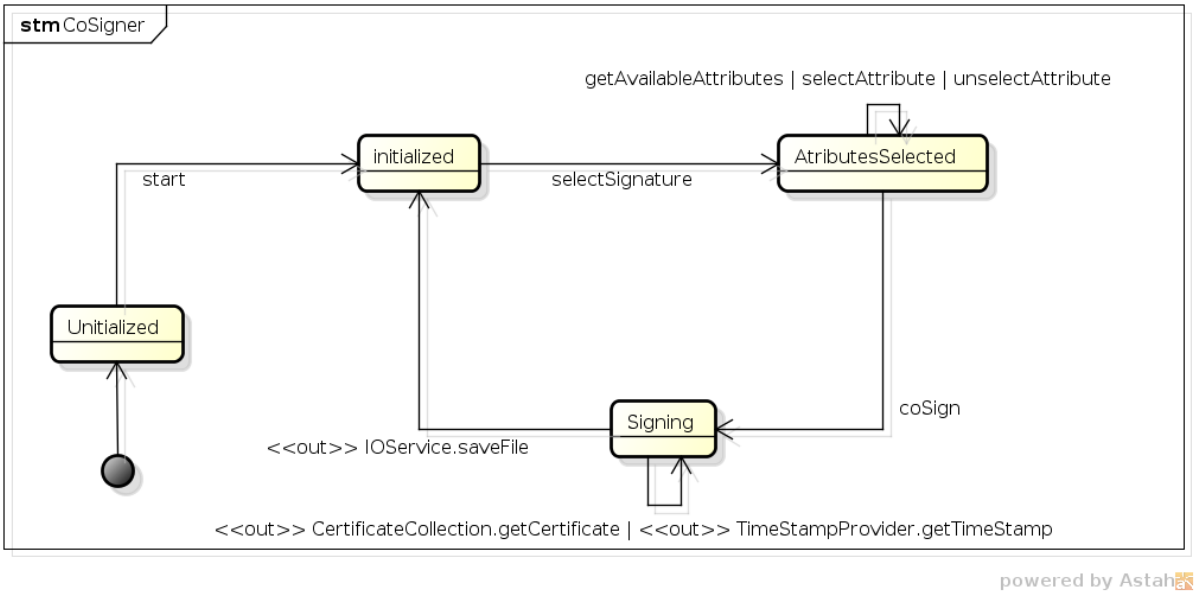

Figura A.2: Diagrama de estados para representar a ordem das chamadas para realizar uma coassinatura.

1.Defina qual o alvo à ser coassinado;

2.Defina a política de assinatura que será utilizada;

Os atributos disponíveis para inclusão na assinatura podem ser obtidos através da chamada *getAvailableAttributes*;

A inclusão de um atributo na assinatura é comandada pela chamada *selectAttribute*.

3.Conclua a assinatura efetuando a chamada *cosign*.

Aqui a configuração da assinatura é verificada;

A assinatura será gerada se a configuração for a correta. As chamadas para obter os dados necessários para a construção de atributos é feita nesse momento;

O componente de assinatura irá chamar o componente *IOGui* através de sua interface para realizar o salvamento da assinatura.

A.3 Contra-Assinar

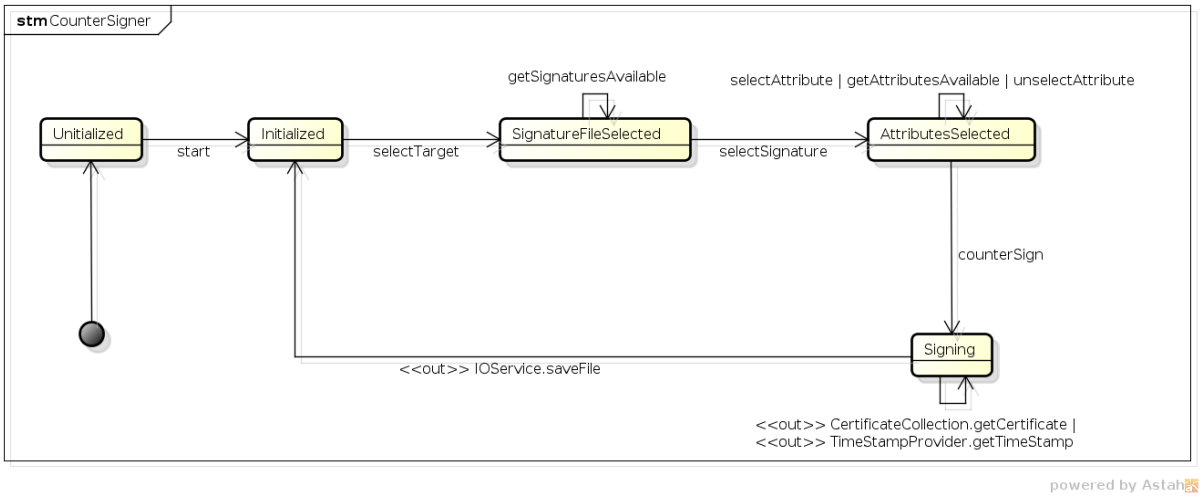

Figura A.3: Diagrama de estados para representar a ordem das chamadas para realizar uma contra-assinatura.

1.Defina o alvo a ser assinado;

2.Defina qual a assinatura presente no alvo que será assinada;

Nesse momento é possível obter uma lista das assinaturas presentes no alvo selecionado.

3.Configure os atributos;

Os atributos disponíveis para inclusão na assinatura podem ser obtidos através da chamada *getAvailableAttributes*;

A inclusão de um atributo na assinatura é comandada pela chamada *selectAttribute*.

4.Conclua a assinatura efetuando a chamada *countersign*.

Aqui a configuração da assinatura é verificada;

A assinatura será gerada se a configuração for a correta. As chamadas para obter os dados necessários para a construção de atributos é feita nesse momento;

O componente de assinatura irá chamar o componente *IOGui* através de sua interface para realizar o salvamento da assinatura.

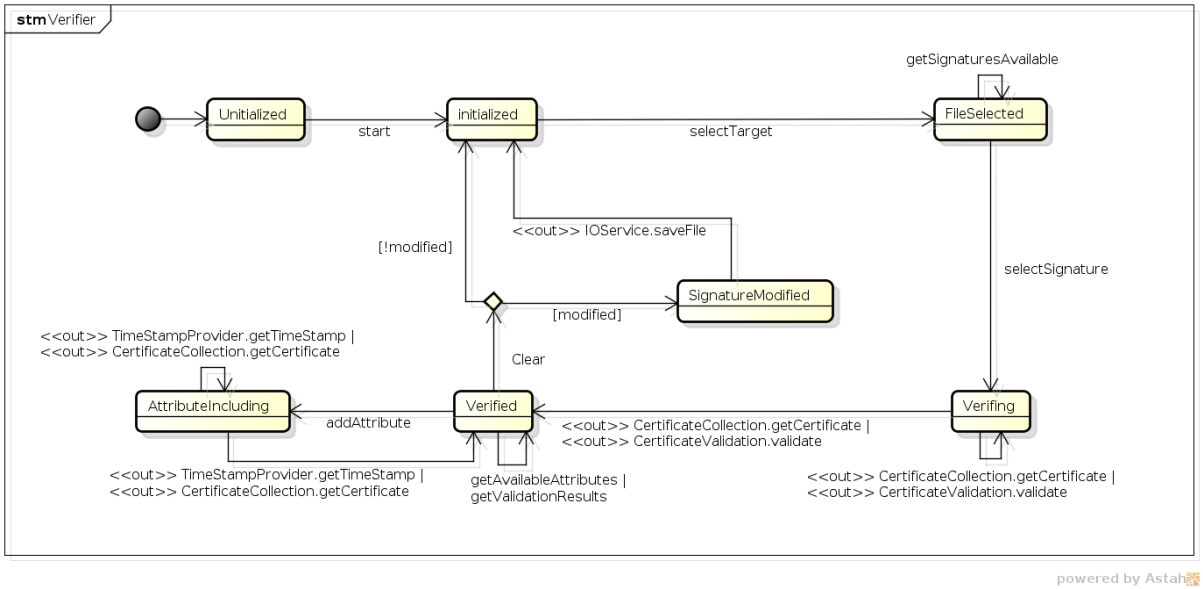

Figura A.4: Diagrama de estados para representar a ordem das chamadas para realizar uma verificação de uma assinatura.

A.4 Verificar

1.Defina o alvo da verificação;

2.Defina qual a assinatura presente no alvo que será verificada;

Nesse momento é possível obter uma lista das assinaturas presentes no alvo selecionado.

3.Execute a verificação da assinatura;

4.Consulte os dados da verificação e/ou adicione atributos na assinatura;

Os atributos disponíveis para inclusão na assinatura podem ser obtidos através da chamada *getAvailableAttributes*;

A inclusão de um atributo na assinatura é comandada pela chamada *selectAttribute*.

5.Encerre a verificação através da chamada *clear*.

Será verificado se aconteceu alguma verificação na assinatura verificada para então efetuar a chamada no componente *IOGui* para salvar a nova versão da assinatura.

A.5 Verificar Caminho de Certificação

1.Construa o caminho de certificação através da chamada *buildCertPath* passando o certifi-

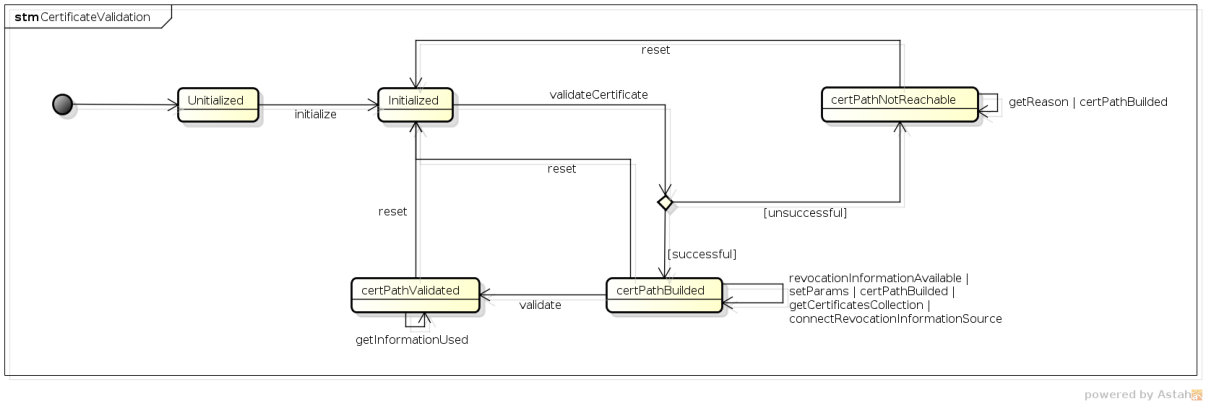

Figura A.5: Diagrama de estados da interação da construção e verificação do caminho de certificação.

cado do qual se quer obter o caminho e as âncoras de confiança.

2. Verifique se o caminho foi construido com sucesso.

   Em caso de erro na construção a razão para o erro pode ser obtida através da chamada *getReason*.

3. Verifique o caminho de certificação.

   Antes da chamada *verify* deve-se definir quais as fontes de dados de revogação que devem ser utilizadas.

4. O resultado da validação do caminho de certificação deve ser obtido através da chamada *isValid*.

   Os dados de revogação utilizados na validação do caminho podem ser obtidos através da chamada *getInformationUsed*.

   Os certificados do caminho podem ser obtidos através da chamada *getCertificates*.

5. Para verificar um novo certificado a chamada *reset* deve ser executada primeiro.

A.6 Obter Chave Privada

1. Para obter a chave privada execute a chamada *getPrivateKey*.

2. O componente irá verificar se o usuário já selecionou alguma entidade e se está já foi autenticada.

   Em caso positivo, a chave correspondente será retornada.

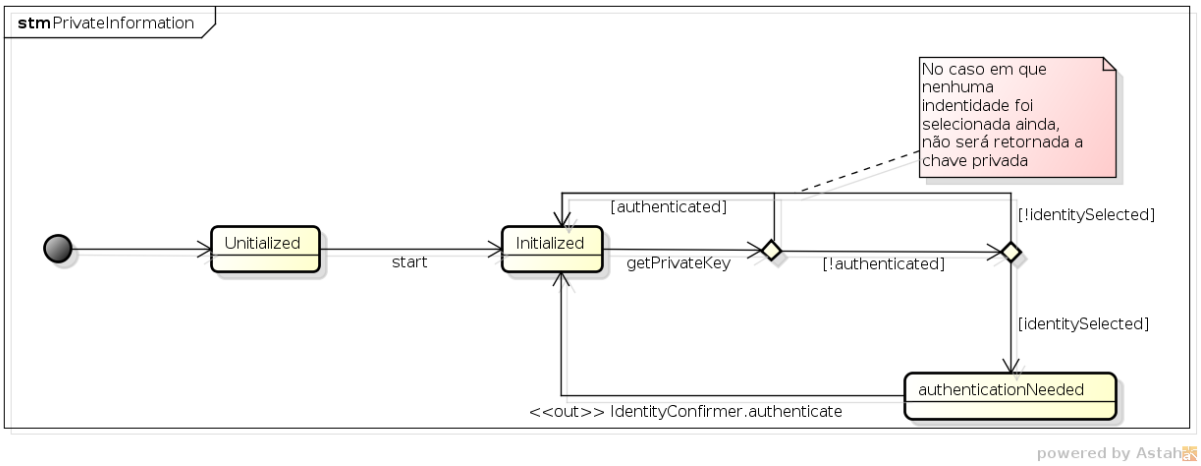

Figura A.6: Diagrama de estados da interação para obtenção de chave privada.

Caso a identidade tenha sido selecionada mas não a autenticação, o componente irá solicitar ao componente IdentityManagerGui a confirmação da identidade.

Caso a identidade não tenha sido selecionada nenhuma chave privada será retornada.

ANEXO B – Arquivo de configuração do Framework de suporte

O arquivo de configuração do *Framework* é um arquivo XML. Ele contém duas estruturas, uma para identificar algum atributo que faz parte da aplicação, outra que indica a conexão entre os componentes da aplicação.

A estrutura que indentifica um componente contém os seguintes dados:

- O nome do componente. Esse nome é o nome completo da classe que extende a classe *Component*, incluindo os pacotes onde essa classe está.
- Uma lista das interfaces que o componente implementa.
- Uma lista das interfaces das quais o componente depende.

A estrutura que identifica a conexão entre os componentes possuí uma lista de conexões. Cada conexão contém as seguintes informações:

- O nome do componente, igual ao utilizado para identificar o componente, do qual há uma dependência.
- O nome do componente, igual ao utilizado para identificar o componente, que implementa a dependência do anterior.
- O nome da interface que caracteriza a dependência.

Segue um exemplo do arquivo de configuração:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<components>
<component name="br.ufsc.labsec.signature.cades.CadesSignature">
```

```
<provides>
<role name="br.ufsc.labsec.signature.Signer" />
<role name="br.ufsc.labsec.signature.Verifier" />
</provides>
<requires>
<role name="br.ufsc.labsec.signature.IOService" />
<role name="br.ufsc.labsec.certificateRepository.PrivateInformation" />
</requires>
</component>

<component name="br.ufsc.labsec.signature.gui.signer.SignerGui">
<provides />
<requires>
<role name="br.ufsc.labsec.signature.Signer" />
</requires>
</component>

<component name="br.ufsc.labsec.signature.gui.verifier.VerifierGui">
<provides />
<requires>
<role name="br.ufsc.labsec.signature.Verifier" />
</requires>
</component>

<component name="br.ufsc.labsec.signature.gui.ioService.IoServiceGui">
<provides>
<role name="br.ufsc.labsec.signature.IOService" />
</provides>
<requires />
</component>

<component
name="br.ufsc.labsec.certificateRepository.pkcs12.Pkcs12Repository">
<provides>
<role name="br.ufsc.labsec.certificateRepository.CertificateCollection" />
<role name="br.ufsc.labsec.certificateRepository.PrivateInformation" />
```

```
<role name="br.ufsc.labsec.certificateRepository.IdentitySelector" />
</provides>
<requires>
<role name="br.ufsc.labsec.certificateRepository.IdentityConfirmer" />
</requires>
</component>

<component
name="br.ufsc.labsec.signature.gui.identityManager.IdentityManagerGui">
<provides>
<role name="br.ufsc.labsec.certificateRepository.IdentityConfirmer" />
</provides>
<requires>
<role name="br.ufsc.labsec.certificateRepository.IdentitySelector" />
</requires>
</component>

<connections>
<connection from="br.ufsc.labsec.signature.gui.signer.SignerGui"
to="br.ufsc.labsec.signature.cades.CadesSignature">
<role name="br.ufsc.labsec.signature.Signer" />
</connection>

<connection from="br.ufsc.labsec.signature.gui.verifier.VerifierGui"
to="br.ufsc.labsec.signature.cades.CadesSignature">
<role name="br.ufsc.labsec.signature.Verifier" />
</connection>

<connection from="br.ufsc.labsec.signature.cades.CadesSignature"
to="br.ufsc.labsec.signature.gui.ioService.IoServiceGui">
<role name="br.ufsc.labsec.signature.IOService" />
</connection>

<connection from="br.ufsc.labsec.signature.cades.CadesSignature"
to="br.ufsc.labsec.certificateRepository.pkcs12.Pkcs12Repository">
<role name="br.ufsc.labsec.certificateRepository.PrivateInformation" />
```

```
</connection>

<connection
from="br.ufsc.labsec.signature.gui.identityManager.IdentityManagerGui"
to="br.ufsc.labsec.certificateRepository.pkcs12.Pkcs12Repository">
<role name="br.ufsc.labsec.certificateRepository.IdentitySelector" />
</connection>

<connection
from="br.ufsc.labsec.certificateRepository.pkcs12.Pkcs12Repository"
to="br.ufsc.labsec.signature.gui.identityManager.IdentityManagerGui">
<role name="br.ufsc.labsec.certificateRepository.IdentityConfirmer" />
</connection>
</connections>
</components>
```

ANEXO C – Artigo

Modelagem de um Software Orientado à Componentes para Assinatura Digital

Maurício Simões de Oliveira
Departamento de Informática e Estatistica – UFSC - Ciência da Computação
mauricio.so@inf.ufsc.br

Abstract. *This work is a evolution of the first design proposed in Implementação do Padrão Brasileiro de Assinatura Digital (SILVEIRA, 2011). The division of the components adopted was based on the standards that PBAD use. To demonstrate the concepts adopted in this work was developed a protorype. The prototype can be a reference for future implementations. The implemetation of the prototype is divided in two parts, the first one is a framework for support of the component oriented paradigm, the second one is a simplified implementation of a component oriented digital signature software. In this work was conclude that the development of a component oriented digital signature software for PBAD is possible.*

Resumo. *Esse trabalho é uma evolução da modelagem apresentado no trabalho Implementação do Padrão Brasileiro de Assinatura Digital (SILVEIRA, 2011). A divisão de componentes adotada tomou como base as especificações dos padrões necessários à implementação do PBAD, dividindo os componentes com base no padrão que cada componente implementa. Como forma de demonstrar os conceitos adotados foi desenvolvido um protótipo. O protótipo ainda serve como um exemplo de referência para futuras implementações. A implementação do protótipo inclui um framework para o suporte da orientação à componentes e a implementação de um software de assinatura digital utilizando esse framework. Concluiu-se com esse trabalho que é viável a implementação de um software de assinatura digital utilizando o paradigma de orientação à componentes.*

1. Introdução

A instituição da ICP-Brasil foi feita através da medida provisória 2200-2. O Padrão Brasileiro de Assinatura Digital (PBAD) foi definido visando normatizar as assinaturas realizadas dentro do âmbito da ICP-Brasil. A normatização das assinaturas feita pelo PBAD institui requisitos mínimos que as assinaturas devem cumprir. O PBAD definiu quais os padrões que devem ser utilizados para realizar as assinaturas e as regras sobre as assinaturas. As políticas de assinatura foram adotadas para permitir as aplicações realizar a verificação de conformância das assinaturas na ICP-Brasil. Para simplificar a distribuição das políticas de assinatura, foi desenvolvido a Lista de Políticas Aprovadas (LPA) que não corresponde com um padrão internacional. A LPA é assinada é permite as aplicações localizarem as políticas de assinatura de forma automática .

1.1 Trabalhos Relacionados

O PBAD foi desenvolvido com o objetivo de popularizar e possibilitar o uso de assinatura digital no país. O software de referência do PBAD software foi desenvolvido em duas partes. Uma das partes é a biblioteca para gerar e validar assinaturas ICP-Brasil, a outra parte é o assinador de referência que implementa um assinador completo utilizando a biblioteca desenvolvida (SILVEIRA, 2011).

2. Metodologia

Nesse trabalho foi utilizada a metodologia de desenvolvimento. O desenvolvimento foi baseado em livros e padrões referentes ao assunto tratado. Foi também necessário buscar manuais e outros documentos referentes as ferramentas utilizadas no desenvolvimento.

2.2 Limitações

Como o PBAD é bastante extenso, bem como os padrões no qual ele é fundamentado, esse trabalho não modela tudo que é necessário para uma implementação completa do PBAD. Apenas o componente de assinatura CMS Advanced Eletronic Signature (CAdES) foi modelado. Dentre as técnologias de acesso e gerência de certificados digitais, apenas foi modelado o componente que suporta PKCS#12. Alguns componentes foram simplificados. O componente CMSSignature não implementa o suporte aos atributos que caracterizaram o padrão CAdES. Portanto o protótipo desenvolvido não é um assinador que está de acordo com o normativo do PBAD.

3. Revisão da Literatura

3.1 Resumo Criptográfico

Funções de resumo criptográfico mapeiam um conteúdo de tamanho arbitrário para um conteúdo de tamanho fixo. Com a posse do resumo criptográfico é impossível determinar qual a mensagem que originou tal resumo. Porém com a mensagem e o resumo criptográfico sempre é possível afirmar que o resumo criptográfico corresponde a mensagem, por isso o resumo criptográfico costuma ser utilizado para garantir que o conteúdo está intacto. Um exemplo seria uma das técnicas de assinatura digital, que consiste em se obter o resumo criptográfico da mensagem e cifrar este chave privada, para então enviar ao destinatário a mensagem e o resumo criptográfico cifrado. Dessa forma o destinatário pode concluir que a mensagem é aquela que o dono da chave privada assinou (HOUSLEY; POLK, 2001).

Existem varias funções de resumo criptográfico, seu cálculo costuma ser computacionalmente simples. Um algoritmo de resumo criptográfico é o Secure Hash Algorithm 1 (SHA-1). Outro algoritmo é o Message-Digest Algorithm 5 (MD5), mas esse teve sua segurança comprometida por tornar-se vulnerável a colisões. (HOUSLEY; POLK, 2001).

3.2 Criptografia

A palavra criptografia tem em sua raiz o significado de segredo. Criptografia pode ser vista como uma técnica para embaralhar a mensagem, e depois desembaralhar essa mensagem para trazê-la ao seu estado original. Esse ato de embaralhar a mensagem pode visar a proteção da mesma, para mantê-la confidencial e na criptografia moderna pode ser utilizada também para descobrir se ela foi alterada desde a sua criação (HOUSLEY; POLK, 2001).

3.2.1 Simétrica

A criptografia simétrica se baseia em um segredo apenas, conhecido pelos interessados na mensagem. Esse segredo é chamado de chave, e é utilizado em conjunto com algum algoritmo para cifrar uma mensagem qualquer e então depois decifrá-la.

A criptografia simétrica é eficaz para o caso do sigilo. Caso um terceiro intercepte a mensagem, este terá que conhecer a chave, que não é enviada junto com a mensagem, para poder interpretá-la. Se o terceiro desconhece a chave, um bom algoritmo de criptografia simétrica deve ser forte o bastante para que seja inviável computacionalmente especular qual a chave utilizada para cifrar a mensagem (HOUSLEY; POLK, 2001).

Algoritmos de criptografia simétrica podem ser divididos em duas categorias, os streams ciphers e os block ciphers. Os stream ciphers operam bit a bit cifrando a mensagem e não necessitam de operações do tipo padding, um algoritmo desse tipo é o RC4. Já block ciphers operam sobre conjuntos de dados chamados blocos, esses blocos são de tamanhos variáveis e dependem do tipo de algoritmo utilizado. Um algoritmo importante desse tipo é o AES.

3.2.2 Assimétrica

A criptografia assimétrica, que também pode ser chamada de criptografia de chave pública, se caracteriza por possuir duas chaves complementares envolvidas, a chave pública e a chave privada. A criptografia assimétrica geralmente requer um poder computacional maior que a criptografia simétrica para executar a cifra.

Como a criptografia simétrica costuma ser mais simples do ponto de vista computacio nal, a criptografia assimétrica costuma ser usada para trocar chaves simétricas, para só então criptografar a mensagem desejada. A criptografia assimétrica simplifica muito o processo de manuseio de chaves por possibilitar reduzir o número de chaves que devem ser armazenadas em um ambiente onde, por exemplo, quer se garantir o sigilo da troca de mensagens (HOUSLEY; POLK, 2001).

Um bom exemplo de criptográfia assimétrica é o RSA, que se basea no problema de fatoração de números primos para garantir a segurança de suas chaves. Outro exmplo é o Digital Signature Algorithm (DSA), que se basea num problema matemático conhecido como curvas elípticas, o qual também tem elevado grau de dificuldade computacional.

3.3 Assinatura Digital

A Criptografia Assimétrica provê a base para assinatura digital. Como a chave pública é capaz de decifrar a mensagem cifrada pela chave privada, é possível afirmar quem é o autor da cifra. A assinatura digital consiste em se utilizando da Criptografia Assimétrica, fornecer uma prova de não-repúdio para que o verificador possa identificar o autor. Essa prova é obtida através de um resumo criptográfico cifrado com a chave privada do algoritmo de Criptografia Assimétrica utilizado no esquema de assinatura digital. Podemos citar, por exemplo, um esquema que utiliza RSA e SHA-1, obtém-se o Resumo Criptográfico da mensagem, para então cifrá-lo com a chave privada. Mensagem e Resumo Criptográfico cifrado são enviados para o interessado, que então, para verificar a assinatura, refaz o processo de obtenção do Resumo Criptográfico e decifra o Resumo Criptográfico recebido com a chave pública do autor. Se os dois resumos forem idênticos, então a mensagem é íntegra e pode-se afirmar que o autor da assinatura é o dono da chave privada correspondente à chave pública utilizada para decifrar o Resumo Criptográfico (HOUSLEY; POLK, 2001).

3.3.1 Padrão Brasileiro de Assinatura Digital (PBAD)

O PBAD visa proporcionar um padrão para o uso de assinaturas digitais no Brasil (ITI, 2010a). Em (ITI, 2010b) são descritos os requisitos do PBAD, esses requisitos são restrições sobre os padrões CAdES e XAdES. O PBAD possue dez perfis de assinatura, cinco para CAdES e cinco para XAdES. Cada perfil define um conjunto de atributos necessários para garantir a válidade de uma assinatura digital durante um determinado período. A identificação do perfil é obrigatório na ICP-Brasil é ele é feito através da identificação da política de assinatura de forma explicita (ITI, 2010c). Em (ITI, 2010c) é introduzido o conceito de LPA.

3.4 Padrões de Projeto

De acordo com (HELM, 1995): Sistemas orientados à objetos possuem formas estruturais recorrentes. Design Patterns(Padrão de Projeto) são a proposta para documentar e representar essas recorrencias de design no projeto de aplicações. O nome de um Design Pattern indica onde este pode ser usado e o seu contexto. Design patterns podem ser aplicados em qualquer tipo de sistema, desde a interface até a persistência de uma aplicação. Um Design Pattern visa resolver um problema universal. Uma grande porção de um sistema é coberta por Design Patterns. Sistemas orientados a objetos podem ser decompostos em subsistemas, e estes subsistemas geralmente são formados por um ou mais Patterns nas suas classes principais. Os patterns relativos aos relacionamentos entre objetos ajudam a definir a arquitetura do sistema.

O nome de um design pattern abstrai e identifica um design de estrutura recorrente. O Design Pattern se aplica a um problema em questão e esse problema descreve as circunstâncias em que o Pattern pode ser aplicado, se este Pattern puder ser aplicado à outros problemas semelhantes, então haverá consequencias e trocas por utilizar-se de um Pattern em um escopo mais abrangente do que o previsto.

3.5 Orientação à Componentes

Uma das abordagens existentes para mitigar a alta complexidade do software é a baseada em componentes. Um componente é uma unidade de composição com interface definida por contrato para posterior integração em algum software. O componente define seu comportamento através de interfaces fornecidas e requiridas. Componentes são partes de um sistema que podem ser combinadas a fim de construir um sistema. Um componente só pode ser substituído por outro se estes estiverem em conformidade, ou seja, se o conjunto de interfaces requeridas e fornecidas forem os mesmos e o comportamento externo dos componentes seja semelhante. (OMG, 2010)

Componentes caracterizam o reuso de código. O desenvolvedor não precisa entender como o componente funciona, ele apenas precisa entender como o componente se comporta. Ao desenvolver um componente, há uma troca entre funcionalidade e simplicidade, quanto mais funcional é um componente, ou seja, quanto mais configurável e mais adaptável é o componente, maior a sua complexidade, portanto será mais difícil entende-lo. Componentes simples, embora muito facilmente entendidos, tem uma aplicação limitada, componentes mais complexos podem ser aplicados em uma gama maior de problemas. (JOHNSON, 1997)

3.6 Frameworks

De acordo com (JOHNSON, 1997): As definições de um framework são vagas, mesmo os frameworks sendo utilizados a muito tempo, ainda não se encontrou uma definição clara do que é um framework. Frameworks podem ser vistos como o esqueleto de uma aplicação, definindo uma estrutura de classes abstratas que modela como as intâncias

dessas classes interagem ou como um esboço de uma aplicação que pode ser customizada pelo desenvolvedor para se tornar uma aplicação final.

O reuso de software na era orientada a objetos teve como primeiro interesse o uso de componentes, e posteriormente o reuso de design na forma de patterns. Frameworks são uma forma intermediaria de reuso, que possibilita tanto o reuso na forma de componente, pois o framework inclui código que provavelmente o usuário nunca irá ver, e sim apenas utilizar, como reuso na forma de design, pois a composição do framework em classes abstratas compele a aplicação que se utiliza do framework a se utilizar de um design previamente pensado. O reuso no framework acontece através de classes abstratas e da inversão de controle.

Frameworks são como componentes, porém extremamente customizáveis e entretanto de difícil apreendizagem. Quando um framework é bem desenvolvido, ele pode ser útil para desenvolver uma grande gama de aplicações e irá reduzir o tempo necessário para o desenvolvimento delas. Frameworks e componentes devem ser vistos como coisas distintas, embora ambos incitem o reuso, o framework está mais para um contexto reusável para componentes. Frameworks também são similares a outras técnicas de reuso de design, porém frameworks são especificados em linguagem de programação, tornando o aprendizado dos programadores mais simples e eficiente. Design patterns também são formas de se reutilizar o design para problemas especificos, eles padronizam a solução e definem um contexto para sua aplicação, porém além disso frameworks são código, e por isso um framework fornece uma forma do desenvolvedor testar se o seu entendimento do framework está correto. Outro fato é que o tempo gasto para implementar o framework será reaproveitado, tornando o desenvolvimento da aplicação mais rápido.

4. Desenvolvimento

O software de referência para assinatura digital no Brasil foi implementado em dois módulos principais, uma biblioteca de assinatura digital e o outro um assinador minimalista (SILVEIRA, 2011). Embora temnham mais módulos tanto no assinador minimalista, quanto na biblioteca de assinatura digital, esses módulos não estão explicitados no projeto. A manutenção do assinador minimalista e da biblioteca constitui uma tarefa complexa, pois toda vez que mudanças ocorrem na interface da biblioteca, as alterações são propagadas por todo o código do assinador minimalista e via de regra por outras partes da biblioteca de assinatura digital. A ideia de separar todo esse conjunto de códigos em componentes surgiu da observação de que, embora existam depêndencias, há um número muito maior de módulos que podem ser isolados de forma a estabilizar e explicitar as interfaces entre eles e assim produzir um software com menor grau de acoplamento.

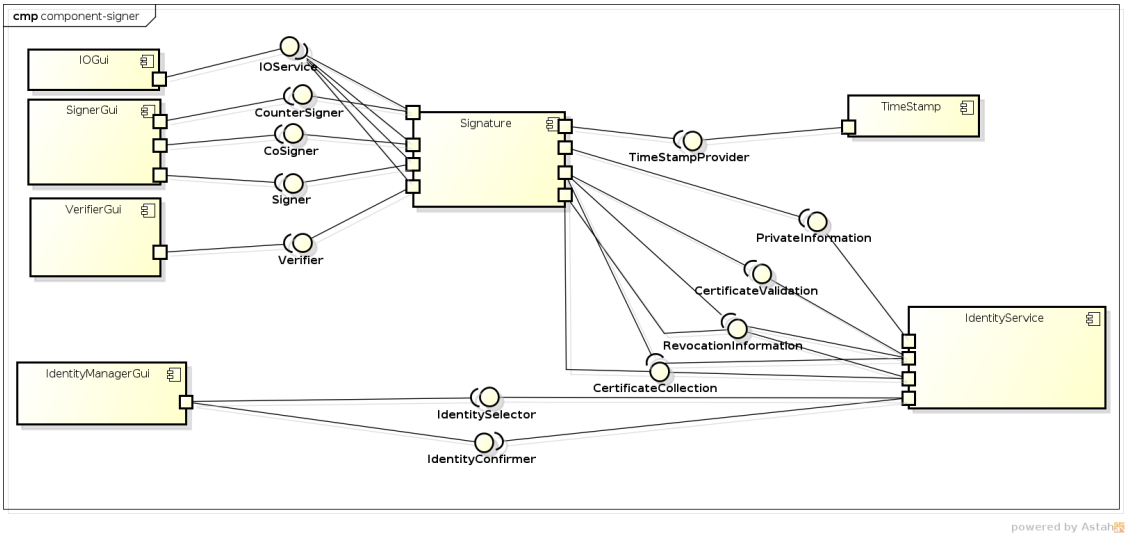

Figura 1. Visão geral dos componentes propostos para implementação do Assinador.

4.1 Design Geral

Para separação de um software em componentes é necessário um critério para poder identificar quais seriam os componentes reutilizáveis dentro de uma organização. Como as partes envolvidas em um software de assinatura digital são padronizadas para propiciar a interoperabilidade, o critério adotado visa separar os componentes segundo a implementação de cada padrão, com isso se espera uma maior possibilidade de reutilização dos componentes. O design do assinador foi também inspirado no atual projeto do padrão brasileiro de assinatura digital. O objetivo é de tornar mais simples a reutilização do código já existente, afim de minizar o tempo de desenvolvimento. Isso também se justifica pois à partir da comparação com softwares (EID) (CUTE) de assinatura digital que implementam os mesmos padrões, observou-se que algumas das soluções adotadas na implementação atual são recorrentes. Essas soluções recorrentes podem indicar que algumas das soluções adotadas no atual projeto são boas soluções para os problemas encontrados.

4.2 Os Componentes do Assinador

O assinador foi dividido em sete componentes, sendo que SignerGui, VerifierGui e IdentityManagerGui são componentes de interface com o usuário. O componente IOGui oferece o suporte do sistema para salvar streams de dados no disco. Os componentes CadesSignature, IdentityService e TimeStamp implementam os padrões necessários a um assinador digital.

4.3 Protótipo

O protótipo visa provar a viabilidade do desenvolvimento de um software orientado à componentes para assinatura digital. Para tal foi desenvolvido um framework que suporta a orientação à componentes e um assinador simplificado capaz de realizar assinaturas CMS. O design do Mini Assinador é uma simplificação do proposto no para o Assinador Orientado à Componentes.

4.3.1 Framework de Suporte

O framework de suporte implementa as noções fundamentais da orientação à componentes. Para tornar possível a composição do software através de componentes o framework fornece uma classe padrão. Essa classe representa um componente que deve ser estendida para criar um componente. A configuração das ligações e dos componentes de um software são feitas através de um arquivo de configuração. O framework é capaz de a partir do arquivo de configuração instanciar os componentes e conecta-los.

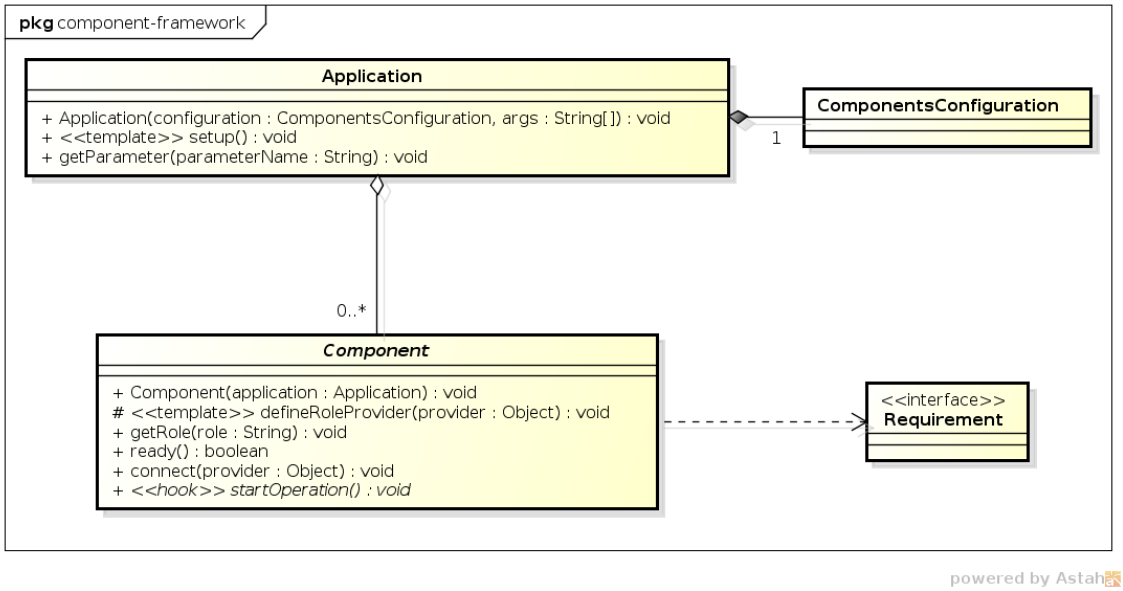

Figura 2. Diagrama de classes do framework de suporte desenvolvido.

4.3.2 Mini Assinador

O Mini Assinador demonstra de forma prática a viabilidade da implementação de um assinador orientado à componentes. Embora o protótipo desenvolvido não implemente

um assinador de acordo com o PBAD, seu propósito era testar o framework desenvolvido e verificar a corretude do design proposto. As simplificações feitas para tornar viável a implementação do Mini Assinador em tempo hábil não implicam em alterações nas comunicações entre os componentes, mas apenas nas funcionalidades disponibilizadas pelos componentes envolvidos.

5. Conclusão

A modelagem proposta nesse trabalho é modular devido a natureza do paradigma de orientação à componentes. Pelo fato da alta modularidade do software o esperado é que as mudanças que ocorrerem sejam mais isoladas e impliquem em menos problemas no software como um todo. Componentes com uma interface bem definida e documentada são amigáveis ao desenvolvedor. Como o comportamento do componente é bem definido é provável que algum desenvolvedor se sinta mais a vontade e entenda mais facilmente essa interface sem se preocupar com os detalhes de implementação.

Para desenvolver o protótipo foi necessário o desenvolvimento de um pequeno framework. Embora o framework propicie uma orientação à componentes de forma mais explícita ele pode limitar as possibilidades de reuso do componente, uma vez que para utilizar os componentes é necessário utilizar também o framework, ou então, reimplementar tudo aquilo que o framework faz.

Uma das principais dificuldades encontradas durante o desenvolvimento do trabalho foi a modelagem de interação entre os componentes. Não há uma maneira de se estabelecer a forma de interação entre os componentes que será a melhor sem ser a experiência no domínio.

5.1 Trabalhos Futuros

Nesse trabalho foi modelado um conjunto de componentes necessários para a implementação do padrão CAdES. Como continuação desse trabalho fica a modelagem dos componentes para implementação do padrão XAdES e dos outros padrões de acesso à informação de identidade.

A modelagem realizada nesse trabalho abrange apenas uma das etapas do desenvolvimento de um software. O processo de desenvolvimento adotado deve levar em consideração o fato do software em desenvolvimento ser orientado à componentes. Um estudo do processo de desenvolvimento orientado à componentes que melhor se adeque aos conceitos abordados aqui é também um ponto em aberto.

Referências

SILVEIRA, L. (2011) Implementação do Padrão Brasileiro de Assinatura Digital — Ciências da Computação,Universidade Federal de Santa Catarina.

2.200-2. MEDIDA PROVISÓRIA No 2.200-2 (2001) Disponível em: <http://www.iti.gov.br/twiki/bin/view/Certificacao/MedidaProvisoria>. Acesso em: 2012-11-22.

CUTE. Disponível em: <http://www.cryptolog.com/en/products/toolkit/cute-electronicsignature>. Acesso em: 2012-11-22.

EID Applet. Disponível em: <http://code.google.com/p/eid-applet>. Acesso em: 2012-11-22.

HOUSLEY, R.; POLK, 2001 T. Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure. 1. ed. [S.l.]: Wiley. ISBN 0471397024.

ITI 2010. Visão Geral Sobre Assinaturas Digitais na ICP-Brasil. v. 2.0. Brasília, Abril 2010. DOC-ICP-15.

ITI 2010. Requisitos Mínimos para Geração e Verificação de Assinaturas Digitais na ICP-Brasil. v.2.0. Brasília, Abril 2010. DOC-ICP-15.01.

ITI 2010. Requisitos Mínimos para Políticas de Assinatura Digital na ICP-Brasil. v.2.0. Brasília, Abril 2010. DOC-ICP-15.03.

JOHNSON, R. E. 1997 Components, frameworks, patterns.