CENTRO UNIVERSITÁRIO DE BRASÍLIA -UniCEUB
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUÍS CARLOS MARTINS LEÃO

ASSINATURA DIGITAL DE ARQUIVO PDF INDEPENDENTE DE AMBIENTE UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Orientador: Prof. MsC Francisco Javier de Obaldía Díaz

Brasília
julho, 2013

LUÍS CARLOS MARTINS LEÃO

ASSINATURA DIGITAL DE ARQUIVO PDF INDEPENDENTE DE AMBIENTE UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. MsC Francisco Javier de Obaldía Díaz.

Brasília

julho, 2013

LUÍS CARLOS MARTINS LEÃO

ASSINATURA DIGITAL DE ARQUIVO PDF INDEPENDENTE DE AMBIENTE UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. MsC Francisco Javier de Obaldía Díaz.

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandes
Coordenador do Curso

Banca Examinadora:

_____

Prof. Francisco Javier de Obaldía Díaz, Mestre.

_____

Prof. Roberto Avila Paldês, Mestre.

_____

Prof. Luís Cláudio Lopes de Araújo, Mestre.

_____

Prof. Fabiano Mariath, Mestre.

Dedico este trabalho e a vitória da minha graduação à minha mãe, Tânia, aos meus avós, Clésia e Benedito (*in memoriam*), ao meu tio, Betan, e à minha noiva, Nathália.

AGRADECIMENTOS

Agradeço:

A minha noiva Nathália por sempre estar ao meu lado, apoiando-me nos momentos mais difíceis que passei, sem perder a ternura e a fé em mim;

A minha mãe Tânia por ter me ensinado a persistir sempre, independentemente das adversidades que fossem impostas pela vida;

O meu Tio Betan por ser meu primeiro professor na vida, por ter me ensinado a manter a calma e a analisar as situações ao meu redor, para, então, encontrar uma solução. Por ter brincado comigo quando eu era criança, utilizando componentes eletrônicos e criando programas de computador;

A minha Avó Clésia por seu carinho, paciência e ternura;

O meu avô Benedito por ter me ensinado que é preciso sempre visualizar duas jogadas à frente do adversário, e não apenas uma; e que todos têm medo, mas apenas alguns sabem domá-lo, e estes que o domam conseguem ir muito além do que qualquer um pode imaginar;

O professor orientador Francisco Javier de Obaldía Díaz, pelo estímulo, pela atenção e pela confiança depositada em mim;

O Diretor de Tecnologia, Heber Lucena, e a Gerente de Projetos, Patricia Gomes, da empresa *Sigma Dataserv S.A.* pela disponibilidade em me fornecer toda a estrutura para implementar os testes deste projeto.

“But I don’t want to go among mad people," Alice remarked.
"Oh, you can’t help that," said the Cat: "we’re all mad here. I’m mad. You’re mad."
"How do you know I’m mad?" said Alice.
"You must be," said the Cat, or you wouldn’t have come here.”

Lewis Caroll, Alice in Wonderland

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ADD – Assinatura Digital Distribuída

AC – Autoridade Certificadora

API – *Application Programming Interface*

AR – Autoridade de Registro

ASN 1 – *Abstract Syntax Notation One*

CASD – Computação Aritmética Simples Distribuída

CGMI – Coordenação Geral de Modernização e Informática

CPU – *Central Processing Unit*

DER – *Distinguished Encoding Rules*

DOS – *Disk Operating System*

ICP – Infraestrutura de Chaves Públicas

IETF – *Internet Engineering Task Force*

IP – *Internet Protocol*

JDK – *Java Development Kit*

LAN – *Local-area Network*

MDIC – Ministério do Desenvolvimento, Indústria e Comércio Exterior

MP – Medida Provisória

PDF – *Portable Document Format*

PKI – *Public Key Infrastructure*

RFC – *Request for Comments*

RMI – *Remote Method Invocation*

SERPRO – Serviço Federal de Processamento de Dados

SSL – *Secure Socket Layer*

TCP – *Transmission Control Protocol*

UUID – *Universally Unique Identifier*

WAN – *Wide-area Network*

XML – *Extensible Markup Language*

RESUMO

As ferramentas existentes no mercado para assinar documentos digitalmente são muito caras e não executam o procedimento de maneira distribuída, o que demanda um tempo elevado quando se deseja assinar diversos documentos. Visando solucionar esse problema, esse projeto teve como objetivo geral especificar, desenvolver e implantar sistema que permita assinar uma quantidade arbitrária de documentos PDF, utilizando computação distribuída, de tal forma que não seja possível determinar de qual máquina os recursos foram utilizados. Desenvolveu-se um protótipo denominado Assinatura Digital Distribuída, o qual fez uso de duas bibliotecas, a Computação Aritmética Simples Distribuída e a *JaxFish*. Para construção das bibliotecas, utilizaram-se os padrões de projeto *proxy, observer, template method, state* e *visitor* e as técnicas de programação concorrente, paralela, serialização e *reflection*. O protótipo foi testado na Sigma Dataserv S.A., empresa que presta serviços terceirizados de informática para o Ministério do Comércio e Indústria Exterior, mais especificamente na Coordenação Geral de Modernização e Informática da referida organização. O teste consistiu da assinatura de 15.000 documentos PDFs, utilizando 15 máquinas, com incremento de 5 em 5 computadores. Verificou-se redução no tempo total de assinatura à medida que mais máquinas foram adicionadas ao sistema. Obteve-se um tempo total de 3.250 segundos com a utilização das 15 máquinas, o que equivale a aproximadamente 55 minutos. No entanto, o tempo total superou o tempo teórico esperado, o que se deveu à infraestrutura de rede e ao tempo para ler e escrever um arquivo no disco rígido do computador.

Palavras Chave: Assinatura Digital. PDF. Computação Distribuída.

ABSTRACT

The tools available in the market to sign documents digitally are very expensive and do not perform the procedure in a distributed manner, which demands a high amount of time when it is necessary to sign various documents. In order to solve this problem, this project aimed at specifying, developing and implementing a system to sign an arbitrary amount of PDF documents, using distributed computing, in a way that it was not possible to determine from what machine the resources came from. A prototype called Distributed Digital Signature was developed, which is composed of two libraries, the Simple Arithmetic Distributed Computing and JaxFish. The design patterns proxy, observer, template method, state and visitor and the techniques of concurrent programming, parallel programming, serialization and reflection were used in order to develop the libraries. The prototype was tested in Dataserv Sigma SA, an organization that provides outsourced IT services to the Ministry of Foreign Trade and Industry, more specifically at the Computer and Modernization General Coordination of the company. The test consisted of signing 15,000 PDF documents using 15 machines, with an increment of 5 on 5 computers. Total signature time was reduced as more machines were added to the system. With all 15 machines the total signature time was of 3,250 seconds, which is equivalent to about 55 minutes. However, the total time exceeded the theoretical time expected, which was due to the network infrastructure and the time to read and write a file on the computer's hard drive.

Key Words: Digital Signature. PDF. Distributed Computing.

CAPÍTULO 1 - INTRODUÇÃO

Este capítulo introdutório aborda de forma sucinta os principais tópicos desenvolvidos nesse trabalho. Inicia-se com a apresentação do problema que fundamentou essa pesquisa, seguido dos objetivos, geral e específicos, os quais se buscou alcançar. Na sequência, são introduzidas as justificativas para realização desse estudo e o escopo do projeto. Por fim, expõe-se brevemente sobre os resultados esperados com a aplicação do modelo proposto.

1.1. Apresentação do Problema

A assinatura é definida por Correa (1999) como o "ato pelo qual o autor de um documento se identifica e manifesta a sua concordância com o conteúdo declarativo dele constante [...]". Dentre as modalidades de assinatura, encontra-se a assinatura digital, que é uma tecnologia, baseada em operações criptográficas, que visa a proporcionar autenticidade e integridade a um determinado documento eletrônico (JF, 2013).

Percebe-se que as ferramentas disponíveis no mercado para assinar digitalmente documentos PDF são muito caras e executam a assinatura dos documentos em lote em uma única máquina, como é o caso do assinador *Adobe Pro XI*. Dessa maneira, quando é necessário assinar um volume grande de documentos, é possível levar dias para finalizar o trabalho. Outro efeito colateral que se nota, além da demora, é a indisponibilidade da máquina utilizada para realizar o procedimento de assinatura digital.

Sabe-se que mesmo os computadores modernos estão sujeitos a limitações físicas e que os supercomputadores – os quais possuem maiores recursos – são muito mais caros que as máquinas convencionais, o que os torna inacessíveis a grande parte dos usuários (TANENBAUM, 2007).

Surge, assim, o problema que essa pesquisa visa a responder: Como assinar digitalmente documentos de maneira ágil e econômica?

1.2. Objetivos do Trabalho

O objetivo geral desta pesquisa trata de especificar, desenvolver e implantar um sistema que permita assinar uma quantidade arbitrária de documentos PDF, utilizando computação distribuída, de tal forma que não seja possível determinar de qual máquina os recursos foram utilizados (transparência).

Assegurar a transparência, conforme será detalhado no Capítulo 2, é um dos objetivos principais dos sistemas distribuídos (TANENBAUM; VAN STEEN, 2006).

Os objetivos específicos que se fazem presentes são: a) Dissertar sobre computação distribuída; b) Criar um software capaz de suprir o problema; c) Testar a aplicação; e (d) Avaliar os resultados atingidos.

1.3. Justificativa e Importância do Trabalho

Devido às limitações físicas dos computadores modernos e à inviabilidade de aquisição de supercomputadores pela maioria dos usuários devido aos valores cobrados, a computação distribuída, que se constitui em um conjunto de computadores independentes que cooperam entre si para fornecer uma visão única, coesa e transparente para seus usuários (TANENBAUM, 2007), surge como ferramenta.

Em outras palavras, uma máquina multiprocessada irá executar a assinatura mais rápido que uma máquina comum de usuário, porém esse computador de ponta de linha custa muito mais caro do que um computador convencional. Dessa forma, entende-se que, se forem utilizadas mais de uma máquina comum, de uso pessoal, que tem o preço mais acessível que um computador de ponta de linha, para assinar a mesma quantidade de documentos, o custo de processamento na máquina executora irá cair consideravelmente, pois, apesar da assinatura ser executada em várias outras máquinas, será a própria rede que os computadores utilizarão para se comunicar, o que diminuirá o tempo de execução total.

Atualmente, no mercado de informática, não foram encontradas ferramentas que realizem o trabalho de assinar documentos de forma distribuída. Pretende-se, com esse projeto, sanar essa lacuna existente. Ao disponibilizar o uso do recurso de várias máquinas somadas, reduzem-se os custos, uma vez que não será necessário adquirir um servidor de alto

poder de processamento, apenas utilizar as máquinas já existentes na empresa, por exemplo, fora do expediente, desonerando, assim, o orçamento e utilizando *hardwares* de que o local já disponha.

Além da justificativa gerencial de redução de tempo de processamento e, consequentemente, de custos, o projeto justifica-se do ponto de vista acadêmico por desenvolver uma aplicação inovadora dentro da temática da computação distribuída, cujos resultados podem contribuir para estudos futuros na área. Ademais, a revisão de literatura e a metodologia que fundamentaram esse estudo serão de fundamental importância para auxiliar pesquisadores cujo interesse esteja voltado para o tema proposto.

1.4. Escopo do Trabalho

O projeto visa a assinar digitalmente documentos, utilizando apenas certificados do tipo A1, os quais foram escolhidos por possuírem menor nível de segurança e poderem ser gerados em qualquer computador, sem a necessidade de comprar qualquer tipo de licença ou *token*, sendo protegidos por uma senha de acesso, sem a qual não é possível realizar qualquer operação relativa à chave privada associada ao certificado (KAZIENKO, 2003).

O escopo do trabalho ficou restrito a arquivos PDF por estes serem os mais utilizados na digitalização de arquivos. No entanto, o projeto não irá gerenciar eletronicamente os documentos, apenas assiná-los de forma distribuída. Além disso, o sistema pode ser executado em qualquer máquina que tenha suporte para o Java.

A solução desenvolvida não fez uso do RMI (*Remote Method Invocation)*, nativo do Java, pelas dificuldades impostas pelo protocolo, uma vez que na máquina remota seria necessário executar duas aplicações – uma desenvolvida pelo programador e outra cedida pela Sun/Oracle *RMIREGISTRY* – e as classes devem ser distribuídas manualmente para que a máquina virtual possa utilizá-las em tempo de execução, o que demonstra a ineficiência do protocolo (MAASEN et al, 2001).

1.5. Resultados Esperados

Os resultados esperados com a implementação do projeto são: a) que o tempo de assinatura dos arquivos varie para menos sempre que uma nova máquina for adicionada ao nó da rede distribuída para executar a tarefa de assinar digitalmente os documentos; e b) que a máquina principal servidora não se torne indisponível em nenhum momento, podendo esta ser utilizada para outras funções, como servir uma página *web*.

1.6. Estrutura do Trabalho

O presente trabalho está dividido em seis Capítulos, intitulados Introdução, Referencial Teórico, Bases Metodológicas para Resolução do Problema, Assinatura Digital Distribuída (ADD), Aplicação Prática da Assinatura Digital Distribuída (ADD) e Considerações Finais.

O Capítulo introdutório, como mencionado anteriormente, aborda o problema e os objetivos que fundamentaram esse estudo, assim como as justificativas para escolha do tema, o escopo do projeto e os resultados esperados com sua implementação.

O segundo Capítulo apresenta as teorias de base que fundamentaram o desenvolvimento do modelo desenvolvido para solucionar o problema de pesquisa, com especial atenção à computação distribuída.

O Capítulo intitulado Bases Metodológicas para Resolução do Problema descreve as técnicas, metodologias e outras ferramentas que serão utilizadas na construção da "Assinatura Digital Distribuída (ADD)", modelo de resolução do problema proposto no Capítulo 4. Este, por sua vez, apresenta o modelo desenvolvido, com descrição detalhada das etapas necessárias para a consecução e implementação do protótipo.

O Capítulo 5 foca-se na aplicação prática da ADD, com descrição dos resultados atingidos e dos custos de implementação. Além disso, faz-se nesse capítulo a apresentação da organização em que se aplicou o modelo e a avaliação global do projeto.

Para finalizar, o Capítulo 6 aborda as considerações finais deste trabalho, com as conclusões dessa pesquisa e sugestões para estudos futuros.

CAPÍTULO 2 – REFERENCIAL TEÓRICO

Visto que o modelo proposto neste projeto visa a solucionar o problema de como assinar digitalmente documentos PDF de maneira ágil e econômica, fez-se necessária a construção de uma revisão de literatura que explorasse o método de sistemas distribuídos e a modalidade de assinatura eletrônica denominada assinatura digital. Este capítulo inicia-se com uma breve descrição de sistemas computacionais e, a seguir, explana-se sobre sistemas distribuídos. Por fim, disserta-se sobre os aspectos que envolvem a assinatura digital.

2.1. Sistemas Computacionais

Leite (2007) define sistema como "um conjunto de elementos interdependentes que realizam operações visando atingir metas especificadas". Os sistemas computacionais, por sua vez, são aqueles que "automatizam ou apoiam a realização de atividades humanas por meio do processamento de informações".

Um sistema computacional é composto por seis elementos básicos (PRESSMAN, 1995 apud LEITE, 2007), quais sejam:

a) *Hardware:* corresponde às partes eletrônicas e mecânicas (rígidas) que possibilitam a existência do *software*, o armazenamento de informações e a interação com o usuário. Exemplos: CPU, memórias primária e secundária, os periféricos, os componentes de rede de computadores.
b) *Software:* [...] parte abstrata do sistema computacional que funciona num *hardware* a partir de instruções codificadas numa linguagem de programação.
c) Informação: coleção de dados organizados e sistematizados, necessários ao desempenho das tarefas e procedimentos [...] componente fundamental nos sistemas baseados em computador. [...] Sistemas processam e armazenam dados que são interpretados como informações pelos usuários por meio da interface.
d) Usuário: [...] operadores que realizam as tarefas e procedimentos.
e) Procedimento ou Tarefa: compreendem as atividades que o sistema realiza ou permite realizar. As tarefas caracterizam a funcionalidade do sistema e devem permitir aos usuários satisfazer as suas metas [...].
f) Documentação: [...] envolve os *manuais de usuário*, que contém informações para o usuário utilizar o sistema que descrevem a sua estrutura e o funcionamento [...].

A figura 2.1 a seguir apresenta um sistema computacional básico:

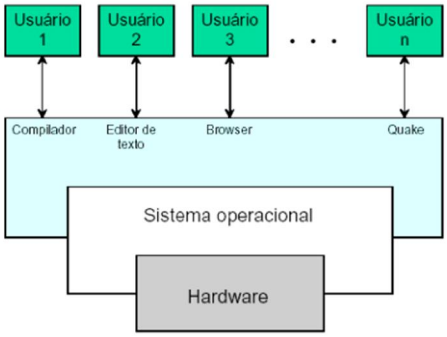

Figura 2.1: Sistema Computacional.
Fonte: OLIVEIRA; CARISSIMI; TOSCANI, 2008.

Sabe-se que, nas últimas décadas, houve a necessidade de se aprimorar os sistemas computacionais, de forma que esses fossem capazes de lidar com a complexidade dos algoritmos envolvidos e com o volume excessivo de dados e informações, o que demanda um alto poder de processamento. Dessa forma, surgiram os sistemas computacionais distribuídos, sobre os quais se disserta na sequência.

2.2. Sistemas Computacionais Distribuídos

Para melhor compreender o modelo de sistema computacional distribuído, esse tópico divide-se em quatro subtópicos, os quais definem conceitos e características dos sistemas, objetivos, tipos, e desafios para a implementação desse modelo.

2.2.1. Conceitos e características

Desde o início da era moderna na computação, em 1945, até aproximadamente 1985, as máquinas eram grandes e caras, o que inviabilizava as empresas de possuírem muitos computadores. Não existiam também redes capazes de conectá-los, e, portanto, essas máquinas operavam de maneira independente (TANENBAUM; VAN STEEN, 2006).

Sabe-se que o surgimento e a disseminação das redes de computadores a partir de meados da década de 80, em especial da *Internet* e das redes que a compõem, foram responsáveis pelo surgimento de um novo paradigma computacional. Trata-se da "possibilidade de distribuição do processamento entre computadores diferentes, [...] [o que]

permite a repartição e a especialização das tarefas computacionais conforme a natureza da função de cada computador" (TANENBAUM; VAN STEEN, 2006).

As *Local-area Networks* (LAN's), redes de computadores de alta velocidade, permitiram que diversas máquinas dentro de um mesmo prédio transferissem pequenas quantidades de informações em microssegundos e grandes quantidades de dados variando entre 100 milhões a 10 bilhões de bits por segundo. Já as *Wide-area Networks* (WAN's) conectaram computadores através do mundo (TANENBAUM; VAN STEEN, 2006).

A revolução pela qual passaram os computadores a partir de meados dos anos oitenta também foi influenciada pelo desenvolvimento dos microprocessadores, os quais possuem a capacidade operacional de um *mainframe*, mas podem ser adquiridos por um valor mais acessível (TANENBAUM; VAN STEEN, 2006).

Nesse contexto, surgiu o conceito de sistema distribuído, definido por Coulousis, Dollimore e Kindberg (2007) como "aquele no qual os componentes de *hardware* ou *software,* localizados em computadores interligados em rede, comunicam-se e coordenam suas ações apenas enviando mensagens entre si".

Tanenbaum e Van Steen (2006), por sua vez, afirmam que existem muitos conceitos para definir "sistemas distribuídos", porém são conflitantes entre si e não comportam todos os aspectos desses sistemas. De forma geral, os autores os definem como "uma coleção de computadores independentes que aparentam ser um sistema único e coerente a seus usuários" (TANENBAUM; VAN STEEN, 2006, tradução nossa).

Os autores levantam, ainda, características importantes dos sistemas distribuídos, sendo a primeira delas que os usuários desconhecem as diferenças entre os computadores utilizados para formar o sistema e as maneiras como estes se comunicam, o que também é válido para a organização interna do sistema (TANENBAUM; VAN STEEN, 2006).

Outra característica relevante apontada pelos autores é que a interação entre os usuários e as aplicações é consistente e uniforme, independentemente do local ou do momento em que estes interajam. Os sistemas distribuídos também podem ser expandidos de maneira relativamente fácil, o que deriva do fato de serem compostos por computadores autônomos. Por esse mesmo motivo, em geral os sistemas continuam disponíveis mesmo que alguns de seus componentes estejam temporariamente fora do ar, e a reposição ou troca de componentes não sejam percebidas pelas aplicações e usuários (TANENBAUM; VAN STEEN, 2006).

Em conformidade com Tanenbaum e Van Steen (2006), Coulouris, Dollimore e Kindberg (2007) também mencionam a facilidade de ampliação do sistema por meio da adição de recursos à rede. Além disso, os autores também apontam como característica dos sistemas distribuídos as falhas independentes, as quais não são imediatamente percebidas pelos demais componentes do sistema (COULOURIS; DOLLIMORE; KINDBERG, 2007).

No entanto, Coulouris, Dollimore e Kindberg (2007) apresentam uma característica negativa não mencionada por Tanenbaum e Van Steen (2006), a qual denominam "inexistência de relógio global". Assim dissertam os autores:

> Quando os programas precisam cooperar, eles coordenam suas ações trocando mensagens. A coordenação frequentemente depende de uma noção compartilhada do tempo em que as ações dos programas ocorrem. Entretanto, verifica-se que existem limites para a precisão com a qual os computadores podem sincronizar seus relógios em uma rede – não existe uma noção global única do tempo correto (...) consequência direto do fato de que a *única* comunicação se dá por meio do envio de mensagens em uma rede (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Com relação à organização dos sistemas distribuídos, Tanenbaum e Van Steen (2006, tradução nossa) explicam que são geralmente organizados de maneira que haja "uma camada intermediária de *software* [*middleware*] logicamente posicionada entre uma camada de alto nível composta por usuários e aplicações e uma camada de baixo nível composta de sistemas operacionais e instalações básicas de comunicações". A figura 2.2 representa a organização de um sistema distribuído.

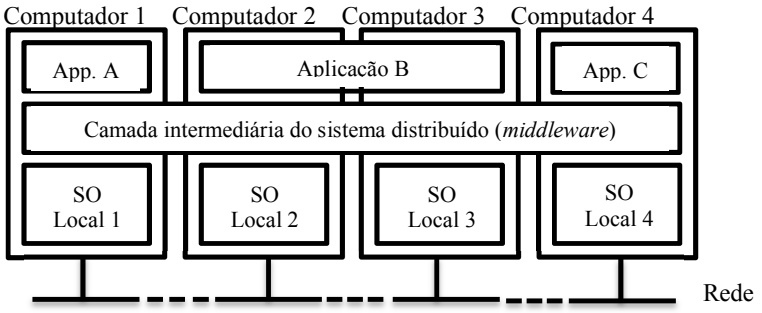

Figura 2.2: Sistema distribuído organizado como camada intermediária, que se estende por múltiplas máquinas e oferece interface única para todas as aplicações**.**
Fonte: TANENBAUM; VAN STEEN, 2006.

No próximo tópico, discutem-se os objetivos expostos por Tanenbaum e Van Steen (2006) para que o desenvolvimento de um sistema distribuído compense o esforço requerido.

2.2.2. Objetivos de um Sistema Distribuído

Quatro objetivos devem ser considerados quando do desenvolvimento de um sistema distribuído, quais sejam: 1) o sistema deve facilitar o acesso aos recursos; 2) os usuários não devem ter conhecimento de que os recursos estão sendo distribuídos através da rede (transparência); 3) o sistema deve ser aberto; e 4) o sistema deve ser escalável (TANENBAUM; VAN STEEN, 2006).

O principal objetivo de um sistema distribuído é permitir o acesso, o controle e o compartilhamento de recursos remotos de maneira eficiente. O motivo fundamental que justifica o compartilhamento de recursos reside na diminuição de custos. Além disso, esse compartilhamento facilita a colaboração entre os usuários e a troca de informações (TANENBAUM; VAN STEEN, 2006).

No entanto, os autores levantam que as facilidades de conectividade e compartilhamento têm a contrapartida de gerar problemas de segurança. Entre esses, encontram-se a falta de proteção contra roubo de informações; a violação de privacidade por meio do uso de informações pessoais para construção de perfis, sem notificação ao usuário; e o envio de comunicações indesejáveis (TANENBAUM; VAN STEEN, 2006).

Outro objetivo básico dos sistemas distribuídos abordado por Tanenbaum e Van Steen (2006) é garantir que os usuários e aplicações não identifiquem que "os recursos e processos do sistema estão fisicamente distribuídos por diversas máquinas", garantindo, assim, a transparência. Os principais aspectos dos sistemas distribuídos a que a transparência aplica-se são evidenciados no quadro 2.3.

Quadro 2.3: Diferentes formas de transparência em um sistema distribuído.

Transparência	Descrição
Acesso	Esconde diferenças na representação de dados e em como um recurso é acessado
Local	Esconde onde o recurso está localizado
Migração	Esconde que um recurso pode ser migrado para outro local
Realocação	Esconde que um recurso por ser realocado enquanto estiver sendo utilizado
Replicação	Esconde que um recurso está replicado
Concorrência	Esconde que um recurso pode ser compartilhado por diversos usuários concorrentes
Falhas	Esconde as falhas e a recuperação de um recurso

Fonte: ISO, 1995 apud TANENBAUM; VAN STEEN, 2006, tradução nossa.

A abertura também é uma meta relevante dos sistemas distribuídos. Nesse sentido, considera-se que um sistema é aberto quando "oferece serviços de acordo com regras padrão

que descrevem a sintaxe e a semântica desses serviços". No caso dos sistemas distribuídos, que normalmente são especificados por meio de interfaces, utiliza-se a Linguagem de Definição de Interface (IDL) (TANENBAUM; VAN STEEN, 2006).

As vantagens de um sistema aberto, explicitadas pelos autores, incluem a facilidade de configuração do sistema por diferentes desenvolvedores e por meio do uso de diferentes componentes e facilidade de adição ou reposição de componentes sem afetar os demais ("um sistema aberto deve ser extensível"). (TANENBAUM; VAN STEEN, 2006).

Por fim, o último objetivo apresentado por Tanenbaum e Van Steen (2006) refere-se à escalabilidade, que pode ser medida de três maneiras diferentes (NEUMAN, 1994 apud TANENBAUM; VAN STEEN, 2006): 1) com relação ao tamanho – possibilidade de se adicionar usuários e recursos ao sistema; 2) geograficamente – possibilidade de se ter usuários e recursos distantes entre si; e 3) administrativamente – facilidade de se administrar, mesmo quando envolve diversas organizações.

Na sequência, discutem-se os principais tipos de sistemas computacionais distribuídos utilizados para se atingir os objetivos supracitados.

2.2.3. Tipos de Sistemas Computacionais Distribuídos

Os sistemas computacionais distribuídos são utilizados para desenvolver atividades de processamento de alto desempenho. De forma geral, subdivide-se em duas categorias: *cluster* e grade computacional (TANENBAUM; VAN STEEN, 2006). Fabieli de Conti (2009) acrescenta, ainda, a categoria de computação em nuvem.

O sistema de *cluster* é utilizado para programação paralela, em que um único programa roda em todos os computadores. Cada máquina corresponde a um nó e todos os nós devem possuir o mesmo sistema operacional instalado e são conectados por meio de uma mesma rede. Ademais, o *cluster* geralmente é composto por equipamentos semelhantes, a fim de diminuir a complexidade do sistema. Portanto, a homogeneidade é característica marcante dessa categoria (TANENBAUM; VAN STEEN, 2006; CONTI, 2009). A figura 2.4 apresenta um exemplo de sistema de *cluster*.

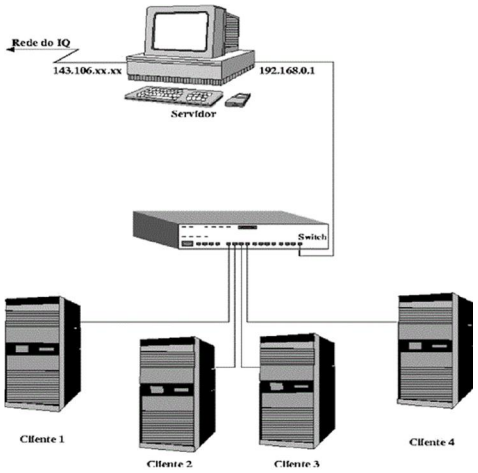

Figura 2.4: Arquitetura em *cluster*.
Fonte: BRAGA, 2002.

Ao contrário dos sistemas de *cluster*, a principal característica de um sistema de grade computacional (*Grid Computing*) é a heterogeneidade (TANENBAUM; VAN STEEN, 2006). De acordo com Foster et al (2002 apud CONTI, 2009), a grade computacional é "uma infraestrutura de *hardware* e *software* que provê acesso seguro, consistente, de forma dispersa e a custo baixo à potencialidade computacional máxima". Nesse sistema, compartilham-se recursos de organizações diferentes, com o intuito de permitir a colaboração entre elas (TANENBAUM; VAN STEEN, 2006). A arquitetura básica de um sistema de grade computacional é representada na figura 2.5 a seguir.

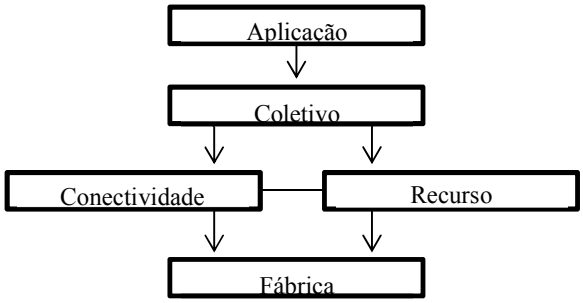

Figura 2.5: Arquitetura em camadas do ambiente de grade.
Fonte: FOSTER et al, 2001 apud TANENBAUM; VAN STEEN, 2006, tradução nossa.

Como se observa na figura 2.5, a arquitetura de uma grade computacional é composta por quatro camadas. Assim as definem Tanenbaum e Van Steen (2006) e Conti (2009):

a) Fábrica: camada que fornece interfaces para operações locais decorrentes das operações de compartilhamento das camadas superiores;

b) Conectividade: camada que define os protocolos de comunicação e autenticação que serão utilizados nas transações;
c) Recursos: camada responsável pelo controle de acesso. Utiliza as funções da camada de conectividade para definir protocolos e API's (*Application Programing Interfaces*) de cada recurso isoladamente;
d) Coletivo: camada que permite o acesso a múltiplos recursos, composta por diversos protocolos que atendem a diferentes propósitos;
e) Aplicação: camada que compreende as aplicações que compõem o ambiente virtual e fazem uso da grade computacional.

A computação em nuvem (*Cloud Computing)*, por sua vez, é definida por Cezar Taurion (2009) como "um conjunto de recursos com capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços disponibilizados na *Internet*". Dessa forma, a "nuvem pode ser vista como o estágio mais evoluído do conceito de virtualização, a virtualização do próprio *data center*". Taurion (2009) ainda acrescenta que a nuvem é uma evolução do sistema de grade computacional.

Para facilitar a comparação entre os três tipos de sistemas distribuídos apresentados neste tópico, o quadro 2.6 resume as principais características de cada categoria:

Quadro 2.6: Características das categorias de sistemas computacionais distribuídos.

Característica	***Cluster***	**Grade Computacional**	**Computação em Nuvens**
Sistema homogêneo	X		
Sistema heterogêneo		X	X
Recursos centralizados	X		
Recursos descentralizados		X	X
Baixa latência	X		
Alta latência		X	X
Rede local	X		
Rede de longa distância		X	X
Aplicações com forte acoplamento	X		
Aplicações com fraco acoplamento		X	
Alta disponibilidade	X		
Compartilhamento de recursos distribuídos		X	

Fonte: CONTI, 2009.

Destaca-se que as três categorias apresentam vantagens e desvantagens, então a opção por uma delas deve refletir os objetivos que se pretende atingir com a implementação do sistema distribuído, além de atender à disponibilidade de recursos do usuário.

À continuação, são apresentados os principais desafios no desenvolvimento de sistemas distribuídos de forma geral.

2.2.4. Desafios para a Implementação de Sistemas Distribuídos

Coulouris, Dollimore e Kindberg (2007) apontam os desafios principais para a implementação de um novo sistema distribuído, relacionados a sete características desses, quais sejam: heterogeneidade, abertura, segurança, escalabilidade, tratamento de falhas, concorrência e transparência.

A heterogeneidade, que está relacionada com o fato de os usuários "executarem aplicativos por meio de um conjunto heterogêneo de computadores e redes", representa um desafio à construção de um sistema distribuído, visto que este deve funcionar em diferentes redes e sistemas operacionais, instalados em diferentes *hardwares*, deve considerar as diferenças entre as linguagens de programação e deve ser desenvolvido com a utilização de padrões comuns de programação. A camada de *software* denominada *middleware* e a migração de código são duas possíveis soluções para este desafio (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Com relação aos sistemas distribuídos abertos, define-se que são

> os sistemas projetados a partir de padrões públicos [...], para reforçar o fato de que eles são extensíveis. Eles podem ser ampliados em nível de *hardware*, pela adição de computadores em uma rede, e em nível de *software*, pela introdução de novos serviços ou pela reimplementação dos antigos, permitindo aos programas aplicativos compartilharem recursos. Uma vantagem adicional [...] é sua independência de fornecedores individuais (COULORIS; DOLLIMORE, KINDBERG, 2007).

Dado que os sistemas abertos são, em geral, desenvolvidos com a utilização de *hardwares* e *softwares* heterogêneos, provenientes de diferentes fornecedores, é necessário que "a compatibilidade de cada componente com o padrão publicado [seja] cuidadosamente testada e verificada", a fim de que o sistema funcione corretamente. As RFC's (*Request for Comments*), documentos que especificam protocolos, são um dos meios utilizados na resolução desse problema (COULOURIS; DOLLIMORE; KINDBERG, 2007).

O terceiro desafio apontado por Coulouris, Dollimore e Kindberg (2007) refere-se à segurança nos sistemas distribuídos. Como os programas se comunicam com outros programas em diferentes máquinas, existem riscos de segurança associados aos recursos de informação que ficam acessíveis na rede. Para sanar tal problema, técnicas como criptografia e instalação de *firewalls* são utilizadas.

Como mencionado anteriormente na seção 2.2.2, a escalabilidade refere-se à possibilidade de se adicionar usuários e recursos ao sistema (TANENBAUM; VAN STEEN, 2006; COULOURIS; DOLLIMORE; KINDBERG, 2007). Os desafios relacionados com a escalabilidade em sistemas distribuídos envolvem: controlar o custo dos recursos envolvidos, controlar a perda de desempenho, impedir que os recursos de *software* esgotem-se e evitar gargalos de desempenho (COULOURIS; DOLLIMORE; KINDBERG, 2007).

O quinto desafio relativo à implementação de sistemas distribuídos relaciona-se com o tratamento de falhas. Sabe-se que as falhas podem gerar resultados indesejáveis ou finalizar um programa antes que este atinja os objetivos pretendidos. Nesse sentido, é vital prevenir e corrigir falhas para garantir a execução correta do sistema. Todavia, Coulouris, Dollimore e Kindberg (2007) explicam que "as falhas em um sistema distribuído são parciais – isto é, alguns componentes falham, enquanto outros continuam funcionando. Portanto, o tratamento de falhas é particularmente difícil". Técnicas utilizadas envolvem detecção de falhas, mascaramento de falhas, tolerância a falhas, recuperação de falhas e redundância de componentes (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Outro obstáculo no desenvolvimento de sistemas distribuídos está relacionado com o compartilhamento de recursos de diferentes aplicativos no sistema, já que pode ocorrer de diversos clientes acessarem um determinado recurso em um mesmo momento. Dessa forma, cabe aso desenvolvedores "[...] fazer o que for necessário para garantir que em um ambiente concorrente [o objeto] não assuma resultados inconsistentes", o que pode ser conseguido, entre outros, pelo uso de técnicas padrões, como semáforos (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Por fim, Coulouris, Dollimore e Kindberg (2007) destacam o desafio de garantir a transparência do sistema, definida como "[...] a ocultação, para um usuário final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes".

Como mencionado na introdução desse Capítulo, para respaldar o modelo proposto para solucionar o problema evidenciado nesse projeto, fez-se necessário compreender o funcionamento dos sistemas distribuídos. Também foi necessário entender a aplicação da assinatura digital, sobre o que se discorre à continuação.

2.3. Assinatura Digital

Miguel Correa (1999) define o termo “assinatura” como o “ato pelo qual o autor de um documento se identifica e manifesta a sua concordância com o conteúdo declarativo dele constante, isto é, o ato de autenticação pelo próprio autor do documento por ele gerado". Com o uso da assinatura, o autor: “revela a sua identidade pessoal de forma inequívoca; manifesta a sua vontade de gerar o documento e emitir as declarações de vontade ou conhecimento dele constantes; e [...] procura preservar a integridade do documento, isto é, a sua inalterabilidade [...]” (CORREA, 1999).

Dentre os tipos de assinatura, Correa (1999) destaca a assinatura autógrafa – inscrição manual do nome do autor em um documento – e a assinatura eletrônica – que deriva de “vários processos técnicos resultantes do processamento de informações por um equipamento informático”. A assinatura eletrônica, por sua vez, compreende quatro modalidades, quais sejam: código secreto, assinatura digitalizada, chave biométrica, e assinatura digital, ou criptográfica (CORREA, 1999).

À continuação, disserta-se com detalhes sobre a assinatura digital, ou criptográfica. As demais modalidades de assinatura eletrônica não serão abordadas, por não terem sido utilizadas no desenvolvimento do modelo que será apresentado no Capítulo 4.

2.3.1. Definição e Propriedades da Assinatura Digital

A assinatura digital é uma tecnologia, baseada em operações criptográficas, que visa a proporcionar autenticidade e integridade a um determinado documento eletrônico (JF, 2013), ou seja, visa a garantir a procedência do documento, ou seja, que esse foi assinado “pela entidade ou pessoa que possui a chave criptográfica utilizada na assinatura” (JF, 2013), e assegurar que esse não sofreu qualquer alteração (JF, 2013; CORREA, 1999).

Em outras palavras, a assinatura digital "consiste num 'selo eletrônico', que é acrescentado a uma mensagem e que é criado através de um sistema criptográfico assimétrico, que gera e atribui ao respectivo titular uma 'chave privada' e uma 'chave pública'" (CORREA, 1999), as quais serão explicadas no tópico 2.3.2.

De acordo com Sterbenz (1996 apud CORREA, 1999), a assinatura digital possui cinco propriedades principais:

> (a) é autêntica, pois prova ao destinatário que o subscritor assinou o documento e que este é uma manifestação da sua vontade;
> (b) não pode ser falsificada, pois prova o fato de o documento ter sido marcado pelo subscritor e não por outra pessoa;
> (c) não pode ser usada de novo: é parte do documento e não pode ser transferida para outro documento;
> (d) impede que o documento seja modificado depois de assinado; e
> (e) não pode ser contestada, por ser uma prova de que o signatário marcou o documento.

Kazienko (2003), em consonância com Sterbenz (1996 apud CORREA, 1999), menciona a autenticidade do conteúdo e do autor da mensagem como propriedade da assinatura digital. No entanto, acrescenta a essa lista o fato de a assinatura digital permitir a verificação da data e do horário em que o documento foi assinado e de ser um meio de dirimir disputas, uma vez que permite a verificação por terceiros (KAZIENKO, 2003).

Na sequência, aborda-se o processo de assinatura digital.

2.3.2. Descrição do processo de assinatura digital

Como mencionado anteriormente, a tecnologia da assinatura digital baseia-se em um par de chaves criptográficas, a chave pública e a privada. A primeira é distribuída livremente dentro do certificado e é utilizada para validar as assinaturas; a chave privada, por sua vez, é mantida pelo titular do certificado e é usada para assinar os arquivos (JF, 2013).

O primeiro estágio para a assinatura digital de um documento é a aplicação de uma função matemática denominada *hash function*. Essa função é a responsável pela garantia da integridade do arquivo, uma vez que alterações posteriores a esse geram alterações na função.

Dessa forma, a função equivale a uma impressão digital do conteúdo do documento a ser assinado e funciona da seguinte maneira (JF, 2013):

> A função *hash* realiza o mapeamento de uma sequência de *bits* (todo arquivo digital é uma sequência de *bits*) de tamanho arbitrário para uma sequência de *bits* de tamanho fixo, menor. O resultado é chamado de *hash* do arquivo. Os algoritmos da função *hash* foram desenvolvidos de tal forma que seja muito difícil encontrar duas mensagens produzindo o mesmo resultado *hash* (resistência à colisão) e que, a partir do *hash* seja impossível reproduzir a sequência que o originou.

No estágio seguinte, o signatário do documento criptografa o *hash* com a sua chave privada, de forma a garantir, além da integridade do documento, a autoria ou autenticidade desse. Também agrega ao arquivo o seu certificado digital, o qual permite verificar a identidade do signatário e a imediata verificação da assinatura digital (JF, 2013). Os certificados digitais serão detalhados no item 2.3.3.

Em momento posterior, uma chave pública é utilizada para descriptografar o *hash*. Destaca-se que "só será possível descriptografar a assinatura, se a chave pública for correspondente à chave privada usada para a assinatura" (JF, 2013). Em outras palavras, assim funciona a criptografia de Chaves Públicas:

> A criptografia de chave pública ou assimétrica permite verificar a autoria de um documento assinado digitalmente, uma vez que só é possível decifrar as informações, cifradas com determinada chave privada, utilizando-se da chave pública correspondente. Os pares de chave são únicos. A chave privada é de posse e responsabilidade exclusiva de seu proprietário. Os certificados digitais são documentos digitais que certificam a posse de um par de chaves por um indivíduo ou instituição. O receptor do "pacote" [documento original + assinatura digital (*hash* criptografado + certificado], inicialmente desempacota o certificado e utiliza as funções de ICP para fazer a verificação da validade do certificado e da cadeia de certificação. Validado o certificado, extrai-se a chave pública desse e aplica-se a assinatura (JF, 2013).

Dessa forma, tão logo essa operação criptográfica se concretize, "estará estabelecida a autoria da assinatura e obtém-se o *hash* do documento" (JF, 2013). Para estabelecer a integridade do documento, deve-se aplicar a função *hash* ao documento original e comparar com a função *hash* obtida do documento assinado.

É relevante destacar que os estágios supracitados compõem um processo realizado por *softwares* de assinatura digital de maneira automática e transparente para os usuários, os quais "emitem avisos caso ocorra falha na validação do documento ou do certificado" (JF, 2013).

Segue-se à dissertação da definição e das propriedades dos certificados digitais.

2.3.3. Certificação Digital

Certificados digitais são documentos assinados digitalmente por uma Autoridade Certificadora (AC) que associam uma chave pública a determinado indivíduo (KAZIENKO, 2003). Conforme Correa (1999), trata-se de

> um documento autêntico, cujo valor deve ser legalmente equiparado ao de um documento notarial e cujo conteúdo deve ser cuidadosamente especificado pela lei, sendo basicamente composto pelo nome e demais elementos de identificação da pessoa do titular, pela chave pública que lhe é atribuída, e pela assinatura digital e chave pública da autoridade certificadora.

Um modelo recomendável de certificado digital é o X.509v3, detalhado na RFC 5280, sobre cujos tópicos relevantes para esse projeto se irá abordar no capítulo 3. O formato X.509v3 provê um serviço de autenticação com estrutura para divulgação e armazenamento de certificados (KAZIENKO, 2003). O quadro 2.7 apresenta a estrutura desse modelo de certificado.

Quadro 2.7: Estrutura do certificado digital X.509 Versão 3.

Versão
Número Serial
Algoritmo de Assinatura
Nome da AC
Período de Validade
Nome do Usuário
Chave Pública do Usuário
Informação Adicional
Informação Adicional
Extensões
Assinatura da AC

Fonte: STA, 1999 apud KAZIENKO, 2003.

Segue-se ao detalhamento da infraestrutura necessária para suportar as operações que envolvam a emissão de certificados digitais.

2.3.4. Infraestrutura de Chaves Públicas

Kazienko (2003) afirma que uma Infraestrutura de Chaves Públicas (ICP) "constitui-se em um conjunto de políticas e procedimentos voltados à operacionalização de um sistema de emissão de certificados digitais baseados em criptografia de chave pública". As ICP's garantem a integridade, a autenticidade e o não repúdio das informações contidas em um certificado digital (KAZIENKO, 2003).

Uma ICP é constituída pelos seguintes elementos (HUN, 2001 apud KAZIENKO, 2003): política de segurança, autoridade certificadora (AC), autoridade de registro (AR), repositório de certificado e sistema de distribuição e aplicações de ICP. Desses, as AC e as AR representam pontos críticos de uma ICP.

A autoridade certificadora é a "entidade independente e legalmente habilitada a exercer as funções de emissão de pares de chaves criptográficas e de dar publicidade às chaves públicas numa lista ou repositório susceptível de ser consultado por qualquer interessado" (CORREA, 1999). Já a autoridade de registro é responsável pela "validação ou autenticação da identidade do usuário e posterior solicitação de emissão do certificado à AC" (KAZIENKO, 2003).

No Brasil, a medida provisória 2002-2, de 8 de agosto de 2001, institui a Infraestrutura de Chaves Públicas Brasileira – ICP-Brasil, com o intuito de "garantir a autenticidade, a integridade e a validade jurídica dos documentos em forma eletrônica, das aplicações de suporte e das aplicações habilitadas que utilizem certificados digitais, bem como a realização de transações eletrônicas seguras" (BRASIL, 2001). A ICP-Brasil é a instituição brasileira responsável pela fiscalização e auditoria dos processos de emissão de certificados digitais.

Os conceitos e aspectos sobre computação distribuída e assinatura digital apresentados nesse Capítulo foram de suma importância para o desenvolvimento da Assinatura Digital Distribuída, modelo proposto para resolução do problema de pesquisa, conforme será evidenciado nos próximos capítulos.

CAPÍTULO 3 – BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

Neste Capítulo, são descritas as técnicas, metodologias e ferramentas utilizadas na formulação do modelo de resolução do problema, modelo este que será apresentado no Capítulo 4. Também são abordadas as justificativas que levaram à escolha de tais métodos e ferramentas.

Inicia-se o capítulo com a explanação das ferramentas de padrão de projetos e as técnicas de programação utilizadas no desenvolvimento da biblioteca CASD, a qual compõe o modelo que será abordado no Capítulo 4. Finaliza-se esse Capítulo com apresentação das metodologias relativas à assinatura digital que foram requeridas para construção da biblioteca *JaxFish*, a qual também será abordada no próximo Capítulo.

3.1. Padrão de Projetos

Padrões são descrições da solução principal de um problema recorrente, a qual pode ser reaplicada em diversas ocasiões de diferentes maneiras (ALEXANDER, 1977 apud GAMMA et al, 2009). Um padrão de projetos expressa soluções em termos de interfaces e objetos e é, em geral, composto por quatro elementos (GAMMA et al, 2009):

a) Nome: identificador que expressa o problema, a solução e as consequências em poucas palavras;
b) Problema: refere-se a quando o padrão deve ser utilizado; descrição do problema a ser solucionado e do contexto que o envolve;
c) Solução: descrição abstrata de um problema que aponta como um conjunto de elementos, de forma geral, pode ser utilizado para solucioná-lo;
d) Consequências: explicam os resultados que se pode atingir com a utilização do padrão e apresentam os dilemas que podem surgir.

De acordo com Gamma et al (2009, tradução nossa), um padrão de projetos “nomeia, resume e identifica os principais aspectos de uma estrutura usual de projetos [...] identifica as classes e instâncias participantes, os seus papeis e suas colaborações, e a sua distribuição de responsabilidades”.

Dentre os tipos de padrão de projetos apresentados pelos autores, dois foram relevantes para o desenvolvimento do modelo apresentado no Capítulo 4, quais sejam: padrões estruturais e padrões comportamentais.

Os padrões estruturais referem-se à maneira como classes e objetos são agrupados para formar estruturas mais complexas. Os padrões estruturais de *classe* estão relacionados com o uso de herança para implementação de interfaces. Já os padrões estruturais de *objeto* relacionam-se com descrições de composições de objetos que criam novas funcionalidades (GAMMA et al, 2009).

Os padrões comportamentais, por sua vez, referem-se a algoritmos e relações de responsabilidades entre objetos. Vão além dos padrões estruturais por descreverem os padrões de comunicação entre classes e objetos. Os padrões comportamentais de *classe* utilizam-se da herança para distribuir comportamentos entre as classes, enquanto os padrões comportamentais de *objeto* utilizam composição. Há, ainda, padrões de *objeto* que encapsulam comportamentos em um objeto para delegar pedidos a ele (GAMMA et al, 2009).

Na sequência, descreve-se sobre os padrões estruturais (*proxy*) e comportamentais (*template method, observer, state* e *visitor*) utilizados no desenvolvimento do modelo proposto neste projeto, a Assinatura Digital Distribuída, com as justificativas para aplicação de tais ferramentas.

3.1.1. *Proxy*

O *proxy* é um padrão estrutural de *objeto* que "fornece um substituto ou um local reservado para outro objeto, a fim de controlar o acesso a ele" (GAMMA et al, 2009, tradução nossa). É utilizado para adiar os custos totais da criação e inicialização de um objeto até o momento em que este seja realmente necessário (GAMMA et al, 2009).

Um *proxy* pode ser aplicado sempre que se necessite de uma referência versátil e mais elaborada para um dado objeto. Situações comuns de uso do padrão *proxy* incluem: *proxy* remoto – fornece um substituto em outro endereço; *proxy* virtual – cria objetos custosos apenas por demanda; *proxy* protetor – controla o acesso ao objeto original; e *smart reference* – substituto de indicador simples que realiza ações adicionais quando o objeto é acessado (GAMMA et al, 2009).

Tendo em vista que um dos objetivos da ADD é oferecer uso transparente de recursos remotos, conforme objetivos dos sistemas distribuídos expressos na seção 2.2.2, entendeu-se que a melhor abordagem seria por meio do padrão de projeto *proxy*, já que os serviços ficam expostos e independem se a instância do objeto encontra-se na máquina local ou remota, ou se é um objeto de um tipo diferente.

Dessa forma, esse padrão permite mascarar a chamada de um método, capturando a intenção da execução desse, e executar em *background* a operação, enviando a requisição para o computador remoto e/ou bloqueando a execução da *thread* até que a resposta do processamento chegue.

3.1.2. *Template Method*

O *template method* é um padrão comportamental de *classe* que "define o esqueleto de um algoritmo em uma operação, submetendo alguns passos a subclasses" (GAMMA et al, 2009, tradução nossa). Esse padrão permite que subclasses redefinam estágios de um algoritmo sem modifica-lo, por meio de operações abstratas que geram comportamentos concretos (GAMMA et al, 2009).

O padrão *template method* deve ser utilizado nas seguintes situações: (a) para implementar as partes fixas de um algoritmo e submeter a subclasses a implementação dos comportamentos variáveis; (b) para evitar duplicação de código, ao manter comportamentos comuns das subclasses em uma classe única; e (c) para controlar a extensão das subclasses (GAMMA et al, 2009).

Conforme apresentado na seção 2.2.2, o principal objetivo dos sistemas distribuídos é facilitar o acesso aos recursos (TANENBAUM; VAN STEEN, 2006). O padrão *template method* foi utilizado para controlar a extensão das subclasses, a fim de atingir esse objetivo.

O padrão permite especializar os eventos que serão ouvidos dentro da ADD, chaveando-os durante o tempo de execução. Ademais, auxilia na definição do método que será implementado e evita *casts* – transformações – desnecessárias. Assim, ao utilizar este padrão, o objeto interessado não precisa executar um *cast* explicitamente, já tendo como retorno ou parâmetro o tipo do objeto desejado.

3.1.3. *Observer*

O padrão comportamental de *objeto* denominado *observer* é utilizado para "definir dependência entre objetos de forma que, quando o *status* de um objeto seja modificado, todos os seus dependentes sejam notificados e atualizados automaticamente". A interação entre os principais objetos utilizados nesse padrão, *subject* e *observer*, é conhecida como *publish-subscribe*. O *subject* envia notificações, sem saber quem são os *observers* dependentes dele, uma vez que diversos *observers* podem subscrever para receber as notificações de um mesmo *subject* (GAMMA et al, 2009, tradução nossa).

As situações que ensejam o uso do padrão *observer* são: (a) quando uma abstração possui dois aspectos, dependentes entre si; (b) quando uma alteração em um objeto qualquer gera a necessidade de alteração em um número de objetos indeterminado; e (c) quando é necessário que um objeto notifique outros, mas sem que saiba quem são esses objetos (GAMMA et al, 2009).

Assim como o padrão *proxy*, o padrão *observer* foi utilizado com a finalidade de garantir transparência no acesso aos recursos da ADD, atingindo um dos objetivos levantados por Tanenbaum e Van Steen (2006). Ademais, o uso desse padrão permite que a quantidade de ouvintes aumente ou diminua conforme disponibilidade, garantindo a escalabilidade (TANENBAUM; VAN STEEN, 2006) do sistema como um todo.

O padrão *observer* foi utilizado em conjunto com o padrão *template* no desenvolvimento da biblioteca CASD, a qual compõe a ADD e será detalhada no próximo Capítulo. Os observadores registram-se ao implementar a forma concreta do tipo de objeto que desejam ouvir. Assim, em conjunto, os dois padrões formam o motor de recebimento dos eventos síncronos e assíncronos.

Esse padrão torna o sistema reativo, poupando processamento e checagem desnecessária da fila de objetos recebidos. Outro ponto forte no uso do padrão *observer* é que a ação sempre será executada na *thread* que disparou o evento. Ou seja, como o contexto de cada entidade responsável por disponibilizar os recursos remotos na biblioteca CASD é executado em uma *thread* especializada, o sistema não irá bloquear para recebimento dos dados.

3.1.4. *State*

O padrão comportamental de *objeto* denominado *state* "permite que um objeto altere seu comportamento quando haja alteração em seu estado interno. A impressão é que o objeto troca de classe" (GAMMA et al, 2009**,** tradução nossa). Nesse padrão, introduz-se uma classe abstrata chamada *TCPstate* para representar todos os estados da rede de conexão, a qual possui uma interface comum a todas as classes que representam estados operacionais distintos. As subclasses da *TCPstate* serão responsáveis por implementar os comportamentos específicos de cada estado (GAMMA et al, 2009).

O padrão de projetos *state* deve ser utilizado em qualquer das seguintes situações: (a) quando o comportamento de um objeto dependa do seu estado e deva ser alterado em *run-time*; ou (b) quando as operações tenham instruções condicionais grandes, compostas por várias partes, as quais dependam do estado do objeto.

O uso do padrão *state* garante a abertura do sistema (TANENBAUM; VAN STEEN, 2006) uma vez que qualquer implementação será enviada automaticamente ao executor remoto, assegurando a extensibilidade da biblioteca CASD. Ademais, o envio automático de definições também contribui para a transparência do sistema (TANENBAUM; VAN STEEN).

Essa ferramenta é fundamental para não enviar repetidas vezes para as entidades executoras das instruções remotas da supracitada biblioteca as definições binárias de uma classe. Por meio desse padrão, o fluxo do código responsável por enviar as requisições e as definições permanece constante, mas o comportamento é alterado após o primeiro envio.

3.1.5. *Visitor*

O *visitor* é um padrão de projetos comportamental de *objeto* que "representa a operação a ser realizada com base em *objectstructure*. Permite que sejam definidas novas operações sem que sejam alteradas as classes dos elementos em que opera" (GAMMA et al, 2009, tradução nossa).

O padrão *visitor* deve ser utilizado em três situações distintas, quais sejam: (a) quando haja necessidade de realizar operações em objetos cujas estruturas possuam muitas classes com interfaces diferentes; (b) quando haja necessidade de realizar operações não relacionadas

em objetos e não se deseje poluir suas classes; e (c) quando se deseje definir com frequência novas operações para uma estrutura cujas classes que definem o objeto raramente sejam alteradas (GAMMA et al, 2009).

O padrão de projetos *visitor* foi utilizado para tornar mais robusto o motor da CASD responsável por receber informações remotas, além de tornar a modificação deste motor desnecessária, já que toda a lógica de recebimento e processamento das informações está condida no *visitor.*

Ao receber uma mensagem remota, ao invés de executar uma sequencia de *if's* e *else's*, o padrão *visitor* permite que a própria classe chaveie para o receptor correto. Assim, o fluxo do programa permanece imutável, independentemente do objeto que estiver em execução.

Somado ao poder de chaveamento em tempo de execução, ao se criar novas mensagens com processamentos específicos, o motor responsável por receber as informações não precisa saber qual ação deve ser tomada, já que cada *visitor* sabe como processar o cliente.

Caso se optasse por não utilizar esse padrão, para cada mensagem que chegasse ao receptor, seria necessário efetuar uma validação de tipo e, para cada novo tipo de mensagem, seria necessário adicionar mais código no motor responsável pela recepção da informação.

À continuação, disserta-se sobre as técnicas de programação usadas no desenvolvimento do modelo proposto nesse projeto.

3.2. Técnicas de programação

As técnicas de programação que foram requeridas para o desenvolvimento da Assinatura Digital Distribuída são apresentadas nesse tópico. Inicia-se com uma breve descrição de programação concorrente e paralela, justificando o uso desses paradigmas. Por fim, disserta-se sobre as técnicas de serialização e reflexão.

3.2.1. Programação Concorrente

Um dos primeiros usos da computação concorrente deu-se na construção de sistemas operacionais, porém atualmente essa técnica é utilizada para desenvolver qualquer tipo de aplicação. Ela surgiu como alternativa à programação sequencial, na qual existe apenas um fluxo de controle (*thread*) (LOPES, 1999).

Um programa concorrente, em contrapartida, é formado "por um conjunto de programas sequenciais que são executados de forma independente um do outro e com possibilidade de comunicação entre si" (LOPES, 1999). Em outras palavras, entende-se que a programação concorrente é um paradigma computacional que permite a execução de diversas *threads* simultaneamente, com uso de recursos compartilhados.

A programação concorrente teve que ser utilizada para que uma *thread* estivesse disponível para receber os dados oriundos de uma máquina remota sem que fosse necessário verificar a todo momento se há *bytes* disponíveis para serem recebidos.

3.2.2. Programação Paralela

A programação paralela é uma "aplicação da programação concorrente dentro de arquitetura de computadores paralelos" (LOPES, 1999) que objetiva "transformar grandes algoritmos complexos em pequenas tarefas que possam ser executadas simultaneamente por vários processadores, reduzindo assim o tempo de processamento [...]" (PITANGA, 2008). Trata-se, portanto, do particionamento das operações, para serem executadas com utilização de recursos paralelos.

A programação paralela assemelha-se aos programas sequencias, mas possui diferenças significativas que representam obstáculos ao desenvolvimento de aplicações utilizando essa técnica (PITANGA, 2008), conforme se depreende da exposição a seguir:

a. A memória é distribuída: os dados da aplicação são distribuídos. Portanto, dados precisam ser subdivididos.
b. As partições de dados podem ser acessadas simultaneamente por diferentes partes da aplicação, com isso os acessos aos dados devem estar sincronizados para evitar condições críticas;

c. A distribuição dos dados pode gerar degradação no desempenho devido às trocas de mensagens, então os programas paralelos devem ser escritos cuidadosamente para a obtenção de desempenho aceitável;
d. Como consequência da distribuição, a própria aplicação deve ser subdividida e distribuída em nodos do sistema, o programador deve então aprender a utilizar técnicas de construções de algoritmos paralelos;
e. A divisão do programa em partes cooperantes pode introduzir um baixo desempenho devido à má atribuição dessas partes aos processadores [...];
f. As distribuições dos dados e das funções tornam o programa de aplicação mais complexo. O processo de programação precisa incorporar etapas de desenvolvimento adicionais para localizar e remover erros de sincronização.

Levando em consideração os desafios apresentados pela programação paralela, esta foi utilizada com o intuito de consumir os recursos de processamento de uma máquina remota e atingir o objetivo principal da biblioteca CASD de executar procedimentos de maneira distribuída. A programação paralela foi utilizada em conjunto com a programação concorrente para evitar o bloqueio da *thread* principal do processo em execução.

3.2.3. Serialização

A serialização é definida como "o processo de conversão de um objeto em um fluxo de *bytes* para armazenar o objeto ou fluxo na memória, em um banco de dados, ou em um arquivo" (MICROSOFT, 2013b). Em outras palavras, "é um processo em que o estado de um objeto e seus metadados (como o nome da classe do objeto e os nomes de seus atributos) são armazenados em um formato binário especial" (IBM, 2011).

A finalidade primordial de utilização dessa técnica de programação é preservar as informações de um objeto, de forma que seja possível reconstituí-lo (desserializar) sempre que necessário (IBM, 2011; MICROSOFT, 2013b). Dessa forma, a serialização permite que o desenvolvedor execute, entre outras, as seguintes ações: "enviar o objeto para um aplicativo remoto por meio de um serviço da *Web*, passando um objeto de um domínio para outro, passando um objeto através de um *firewall* como uma cadeia de caracteres XML, ou manter a segurança ou informações específicas de usuário entre aplicativos" (MICROSOFT, 2013).

A serialização foi utilizada para transformar os objetos que deverão ser executados remotamente na biblioteca CASD, para enviar a definição binária de uma classe e a solicitação de execução de um método específico de um objeto. Por meio da serialização, é

possível transformar todo objeto em uma sequencia de *bytes,* enviá-los pela rede e remontá-los como objetos no receptor.

3.2.4. Reflexão

A reflexão computacional (*reflection*) refere-se à capacidade de "um sistema fazer computações sobre si mesmo com o objetivo de alterar sistemas de forma dinâmica" (PAVAN, 2000). Em geral, essa técnica é usada "por programas que requerem a habilidade de examinar ou modificar o comportamento de tempo de execução de aplicação que rodam na máquina virtual do JAVA" (ORACLE, 2013, tradução nossa). Trata-se de uma técnica complexa a qual viabiliza operações que não seriam possíveis de outra maneira (ORACLE, 2013).

De acordo com a Microsoft (2013a), a reflexão é uma técnica a ser utilizada em quatro situações: "(a) quando se tem que acessar atributos nos metadados do programa; (b) para examinar e instanciar tipos em um conjunto; (c) para criar novos tipos em tempo de execução; e (d) para executar ligação tardia, acessar métodos em tipos criados em tempo de execução".

A reflexão foi amplamente utilizada para remontar um objeto do lado do executor após este ter sido enviado pelo solicitador da execução. Após recebimento dos dados serializados, a reflexão é utilizada para, a partir de uma sequencia de *bytes,* obter-se o objeto concreto, conforme solicitado pelo executor remoto.

3.3. Assinatura Digital

Para construção da biblioteca *JaxFish*, o assinador digital de documentos PDF do modelo proposto nesse projeto, foi necessário o estudo de a aplicação de metodologias relacionadas à temática "assinatura digital". Nesse item, apresentam-se os conceitos e principais definições da RFC 5280 e do PDF *Reference* 1.7. Também se descreve brevemente a biblioteca *Bouncy Castle*, a qual compôs a biblioteca *JaxFish*, segundo módulo que compõe a ADD, o qual também será detalhado no Capítulo 4.

3.3.1. RFC 5280

Request for Comments (RFC's) são documentos produzidos pela *Internet Engineering Task Force* (IETF) que especificam protocolos ou tecnologias. Não são normas estabelecidas, visto que existe a possibilidade de serem revistas a partir de comentários ao documento (D'ÁVILA, 2009).

A RFC 5280, a qual especifica a versão três do formato de certificado X509 e a versão dois da lista de revogação do mesmo certificado para uso na internet (COOPER et al, 2008), que são descritos em detalhes, foi estudada com o intuito de compreender o funcionamento da Infraestrutura de Chaves Públicas (*Public Key Infrastructure – PKI*) para obter-se o domínio dos elementos que constituem os certificados digitais, a fim de subsidiar o desenvolvimento da biblioteca especializada em assinar digitalmente documentos PDF, a qual será detalhada no Capítulo 4.

Essa RFC define as regras de formatação de um certificado digital, além de como deve ser efetuado o cálculo que representa a assinatura de um conjunto de *bytes*. Também define como proceder para verificar se o certificado está válido, ou se foi revogado (COOPER et al, 2008).

De acordo com a supracitada RFC (COOPER et al, 2008, tradução nossa),

> o objetivo da Infraestrutura de Chaves Públicas (ICP) é atender às necessidades de identificação automatizada determinista, autenticação, controle de acesso, e funções de autorização. Suporte a esses serviços é o que determinará os atributos contidos no certificado, bem como as informações de controle auxiliares [...] como política de dados e restrições [...].

Para o desenvolvimento da biblioteca já mencionada, também se fez necessário compreender a definição binária dos certificados digitais. Assim, segundo a RFC 5280, um certificado é uma sequência de três campos obrigatórios, *tbCertificate, signatureAlgorithm,* e *signatureValue*. O primeiro campo contém, entre outras informações, "o nome do sujeito e do emissor, uma chave pública associada ao sujeito e um período válido". O segundo campo apresenta "o identificador para o algoritmo criptográfico utilizado pela autoridade do

certificado para assina-lo". Por fim, o terceiro campo contém "uma assinatura digital calculada sobre o ASN 1 DER codificado no campo *tbsCertificate*" (COOPER et al, 2008).

3.3.2. PDF *Reference* 1.7

O documento *Portable Document Format (PDF) Reference* 1.7 equivale ao ISO 32000-1 e contém toda a especificação de um arquivo PDF, ou seja, todas as informações necessárias para se criar um arquivo PDF e sobre as operações que podem ser realizadas nesse arquivo, como assinar digitalmente, criptografar, bloquear, editar, adicionar novas informações, entre outras (ADOBE, 2006).

A seção 12.8 do documento PDF *Reference* 1.7 apresenta as definições de assinatura digital em arquivos PDF e foi estudada com o intuito de entender os requisitos para aplicação da assinatura, uma vez que o modelo proposto nesse projeto tem o objetivo de assinar digitalmente arquivos PDF.

Para se compreender o processo de assinatura, é necessário conhecer a estrutura de um arquivo PDF, passando por todos os elementos que o definem. Somente após estudo deste documento é possível entender que todo arquivo PDF é composto de objetos, dicionários, referências e uma ou várias tabelas de *cross-reference*, dependendo de quantas modificações o arquivo tenha sofrido. Ademais, aprende-se que todo documento PDF é incremental, ou seja, pode-se sempre adicionar ou remover conteúdos por meio do uso dos objetos que são definidos no PDF *Reference* 1.7 (ADOBE, 2006).

A assinatura digital de um documento PDF não é determinada apenas pelos *bytes* que formam o arquivo; na verdade, é necessário passar por todas as estruturas do arquivo, baseado na sequência contida nas tabelas de *cross-reference,* e calcular a assinatura de acordo com o algoritmo selecionado (ADOBE, 2006).

Depois de calculada a assinatura, faz-se necessário adicionar um novo dicionário ao arquivo, um novo objeto contendo o *hash* gerado a partir do conteúdo e uma nova tabela de *cross-reference*. O objeto que contêm a assinatura deverá possuir também a faixa de *bytes* que será assinada a fim de manter a rastreabilidade sempre que houver uma modificação no arquivo e uma nova assinatura for efetuada. Assim, cada assinatura é valida para determinado trecho do arquivo (ADOBE, 2006).

Além disso, o estudo do PDF *Reference* 1.7 deveu-se à necessidade de entender como embutir em um documento PDF a chave pública do assinante, juntamente com as chaves públicas de todos aqueles que estão hierarquicamente acima dele. Essas informações devem ser incluídas no arquivo PDF para que seja possível validar a assinatura e verificar a validade e autenticidade do certificado digital no momento em que o documento PDF for aberto pelo usuário.

3.3.3. Biblioteca *Bouncy Castle*

A biblioteca *Bouncy Castle* é um conjunto de algoritmos usados para cifrar, decifrar e utilizar certificados digitais (BOUNCY, 2013). Foi utilizada na construção da biblioteca *JaxFish*, que compõe o modelo de Assinatura Digital Distribuída, a fim de que não fosse necessário reescrever, nem recriar os algoritmos já disponibilizados pela *Bouncy Castle*.

Como já mencionado na seção 2.3.2, a assinatura digital consiste de um cálculo executado em cima de uma sequência de *bytes*. Os algoritmos são de domínio público e podem ser obtidos nas RFCs que os definem.

A *Bouncy Castle* disponibiliza a implementação destes algoritmos, sem a necessidade de pagar direitos autorais. Trata-se de uma biblioteca que pode ser usada por qualquer pessoa com o intuito de copiar, modificar, distribuir ou vender *softwares* que requeiram a sua utilização (BOUNCY, 2013).

À continuação, será descrito o modelo de Assinatura Digital Distribuída, desenvolvido como forma de atingir o objetivo geral desse estudo. As ferramentas, técnicas e metodologias aqui evidenciadas foram necessárias à construção das bibliotecas que compõe o modelo, conforme se depreende da leitura do Capítulo 4.

CAPÍTULO 4 – ASSINATURA DIGITAL DISTRIBUÍDA (ADD)

Este Capítulo apresenta o protótipo para o modelo de resolução do problema desenvolvido, denominado “Assinatura Digital Distribuída (ADD)”. Inicia-se o Capítulo com uma apresentação geral das bibliotecas desenvolvidas para resolução do problema, seguida da descrição detalhada das etapas requeridas para o desenvolvimento do modelo e para a implementação deste.

4.1. Apresentação Geral dos Componentes da Assinatura Digital Distribuída

A ADD é composta pela junção das bibliotecas CASD e *JaxFish*, ambas desenvolvidas pelo autor, que apresenta uma interface mínima compatível com o objetivo do projeto proposto. A topologia da aplicação é representada na figura 4.1.

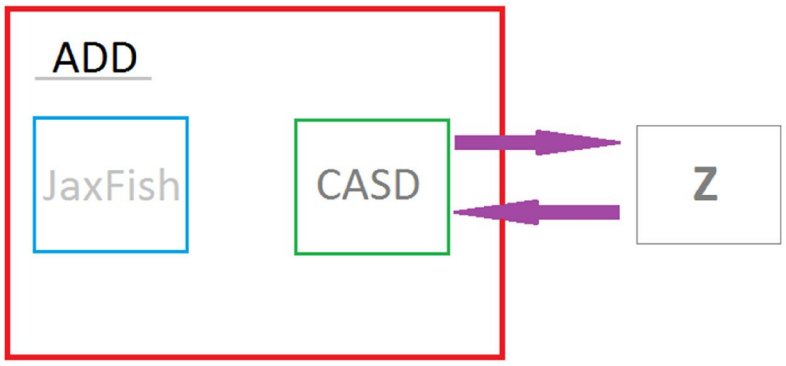

Figura 4.1: Topologia da Assinatura Digital Distribuída.
Fonte: Autor.

A ADD utiliza-se da *JaxFish* para assinar digitalmente documentos PDF. Por meio da CASD, solicita que essa execução seja feita em um ente remoto, caracterizado pelo “Z”, conforme figura apresentada acima. A parte visual da tela da ADD é mostrada na figura 4.2.

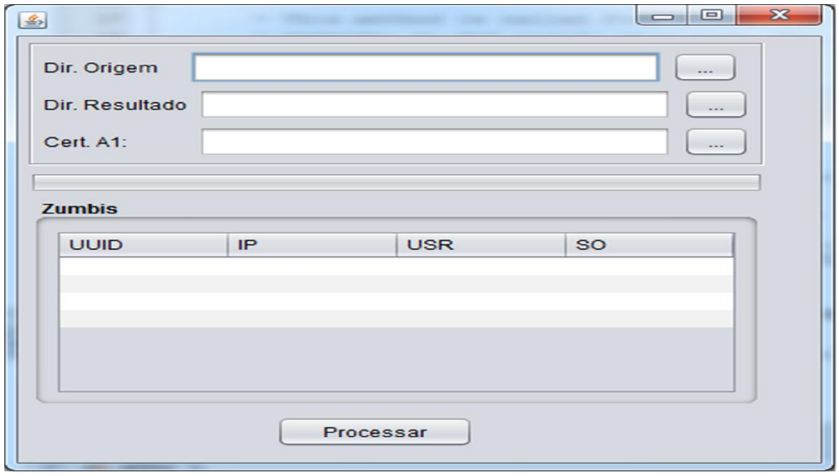

Figura 4.2: Tela da ADD.
Fonte: Autor.

A tela acima contempla apenas o diretório que contêm os arquivos PDF que deverão ser assinados, o diretório em que os arquivos serão salvos depois de efetuada a assinatura e o caminho para o certificado digital tipo A1.

Conforme novas entidades responsáveis por executar ações remotas na CASD – os zumbis, conforme será explicado no próximo tópico – forem aparecendo na rede, elas serão exibidas na área da *grid* zumbis, expondo os dados específicos de cada uma, como o IP e o UUID.

Segue-se ao detalhamento das bibliotecas que compõem a ADD.

4.1.1. Biblioteca “Computação Aritmética Simples Distribuída (CASD)”

A biblioteca CASD é uma alternativa ao Java RMI, com a vantagem de que o programador usuário não é obrigado a herdar direta ou indiretamente de *RemoteObject*. Como o Java não possui herança múltipla – uma classe pode ser filha apenas de uma única classe –, os objetos não podem ter uma hierarquia de classes concretas ou abstratas definidas arbitrariamente.

Para uso de um objeto remoto pela biblioteca, são obrigatórias as seguintes ações: (a) implementar uma interface que exponha os serviços que podem ser executados de maneira remota e (b) instanciar o objeto, com utilização do método estático *Necromancer*::*undeadInstance*, o qual irá criar um *proxy* local e uma nova instância na

máquina remota. Toda vez que um método do objeto criado for executado, a chamada é capturada e envia-se uma mensagem síncrona com confirmação ao executor remoto, para que este execute o método com os parâmetros solicitados. Dessa forma, a execução local fica suspensa até que o processamento remoto retorne.

Existem duas entidades que possuem papéis específicos e distintos na biblioteca: o Zumbi e o *Necromancer*[1]. Um *Necromancer* pode estar conectado a vários Zumbis, mas um Zumbi somente pode estar conectado a um *Necromancer*. A conexão entre as entidades ocorre sob o protocolo TCP/IP e flui sempre partindo da requisição do *Necromancer* para o Zumbi, conforme mostrado na figura 4.3.

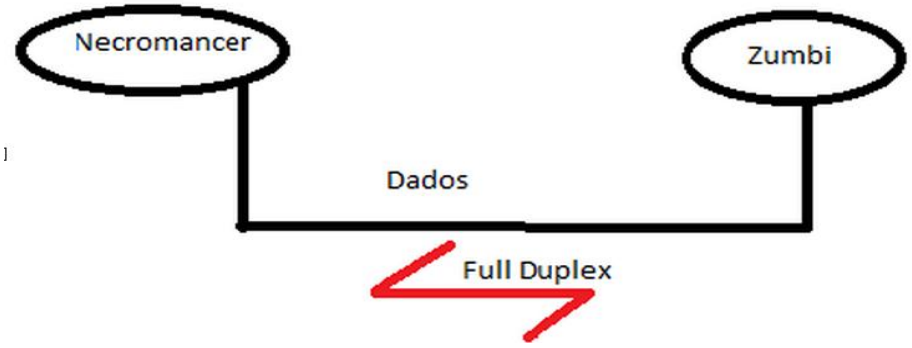

Figura 4.3: Conexão entre as entidade *Necromancer* e Zumbi.
Fonte: Autor.

A entidade Zumbi constitui-se em um executor burro, ou seja, ele apenas é capaz de executar as ordens que recebe – exemplo: execute o método *foo* da classe *bar*, ou receba o dado binário que define a classe *bar.* Por outro lado, o *Necromancer* é capaz de analisar uma classe e gerar informações recursivas sobre as dependências e exceções dela, bem como enviar seus *bytecodes* a um executor remoto para que os procedimentos possam ser executados. Após receber os binários gerados pela *Necromancer*, o Zumbi tem essas novas informações carregadas forçosamente no *ClassLoader* vigente.

O *Necromancer* necessita receber as conexões dos Zombies que estão na rede, após o que cria um contexto interno, gerenciado por uma *thread,* para atender as informações remotas recebidas de forma não bloqueante. As requisições de conexão são recebidas na

[1] Os nomes das entidades foram adotados com base no jogo de tabuleiro AD&D (*Advanced Dungeons & Dragons*), no qual existe a classe de prestígio *Necromancer* que emprega o domínio da arte de reanimar os mortos para satisfazerem a sua vontade. No jogo de fantasia, um *Necromancer* pode ter vários escravos zumbis e um zumbi pode ter apenas um mestre *Necromancer*.

classe interna de *ZombieManager*, que, então, delega a conexão ao *ZombieManager* para registrar o novo Zumbi após a criação de seu contexto.

Um Zumbi possui exatamente um contexto para encapsular e otimizar o envio e a chegada de dados remotos. O *ZombieContext* cria dois canais *bufferizados*, um de entrada e o outro de saída, especializados no envio e recebimento de objetos serializados, respectivamente.

Como cada contexto é uma *thread*, a recepção dos dados bloqueia a execução desta *thread* até a chegada de um novo objeto no canal de entrada e torna a aplicação proativa na detecção de queda na conexão. Assim, no momento em que a conexão entre o *Necromancer* e o Zumbi é estabelecida, uma exceção é gerada e todos os interessados no evento são informados imediatamente. Como as requisições de envio para o Zumbi são executadas pelo usuário, o envio da informação não está condicionado ao desbloqueio da t*hread* de recepção, pois a *thread* à qual o usuário pertence e que solicitou os eventos será a encarregada de enviar as requisições. Dessa forma, o sistema funciona de forma robusta e escalável.

A topologia de funcionamento da biblioteca CASD é monstrada na figura 4.4, em que o "N" representa o *Necromancer* e "Z", os zumbis que estão escravizados ao *Necromancer*.

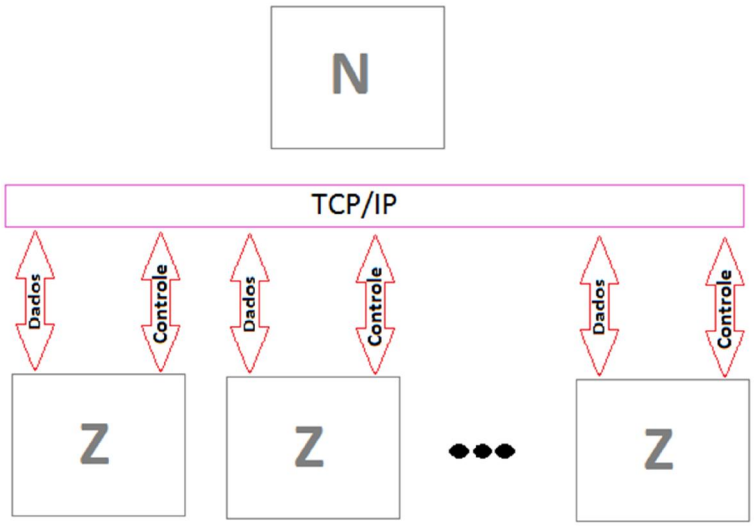

Figura 4.4: Topologia da biblioteca CASD.
Fonte: Autor.

Conforme se depreende da visualização da figura 4.4, um *Necromancer* pode ter vários zumbis ligado a ele. As setas bidirecionais rotuladas de "dados" e "controle" indicam o

fluxo das informações e possuem papeis distintos dentro da aplicação. O canal de dados é usado para requerer execuções remotas, envio de definições das classes e retorno do processamento. O canal de controle, por sua vez, serve exclusivamente para acessar arquivos que estão presentes fisicamente apenas no *Necromancer*.

Todos os pacotes recebidos são transformados na interface *IMessage*, que é a base de todas as partículas de comunicação entre o Zumbie e o *Necromancer*. Após a transformação, a *IMessage* é submetida a um *visitor* – criado a partir do padrão de projeto *visitor* –, o qual irá redirecionar a mensagem aos seus destinatários corretos e, caso seja necessário, retornar ao requisitor uma mensagem de sucesso, ou uma exceção para ser tratada na outra ponta.

Todos os objetos que possuem interesse em receber as informações de um Zumbi devem implementar a interface *DataListener* e adicionarem-se à lista de observadores do Zumbi por meio do método *Zombie::addDataListener*. Desta forma, se houver por alguma razão o interesse de saber quantas requisições foram recebidas de cada Zumbir, o contador implementaria a interface, faria o seu registro no Zumbi e, para cada chamada realizada remotamente, incrementaria o seu contador interno.

Abaixo, apresenta-se um exemplo de como os cálculos são realizados na CASD de forma remota. Ressalta-se que, para executar uma operação remota, é necessário que o código binário correspondente esteja disponível na máquina que executará os procedimentos.

Adota-se arbitrariamente uma condição inicial de um procedimento a ser executado a partir de uma classe chamada SomaSimples. Utiliza-se um método chamado soma, o qual recebe dois parâmetros inteiros e retorna um texto, cuja regra de formação é o texto “Soma:”, concatenado com o resultado da soma dos dois parâmetros inteiros passados. A figura 4.5 mostra o diagrama de classe da classe SomaSimples.

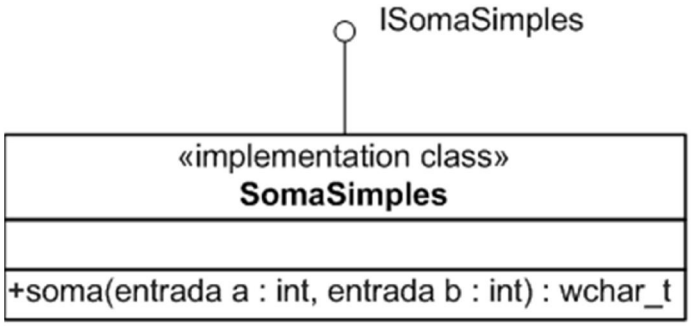

Figura 4.5: Diagrama de Classe da classe SomaSimples.
Fonte: Autor.

Para tanto, considera-se que o pseudocódigo que será executado pelo método soma da classe SomaSimples seja definido como:

"*retorno : literal*

resultado : inteiro

resultado ⬅ *a + b*

retorno ⬅ *('Soma: ', resultado)*"

Após definição da classe e da implementação do método, o código será compilado, gerando um arquivo binário que contem instruções de baixo nível que representam fielmente o resultado esperado no pseudocódigo.

Para execução desse experimento, supõe-se a disponibilidade de 4 máquinas com os endereços IPs 192.168.0.10, 192.168.0.15, 192.168.0.17 e 192.168.0.7 e Sistemas Operacionais Windows 7, Ubuntu 12.04 e MacOS, conforme ilustrado na figura 4.6.

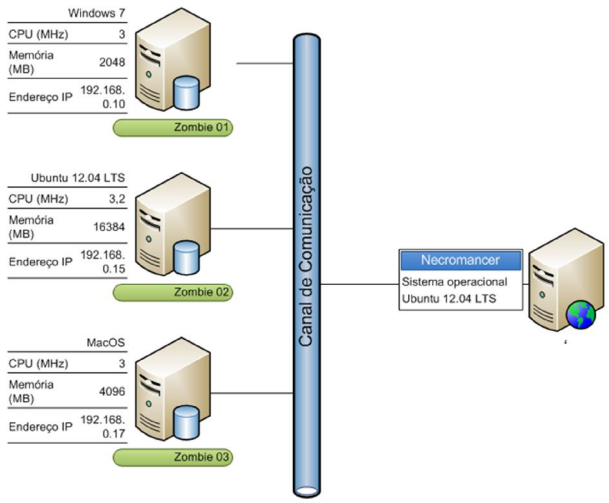

Figura 4.6 Máquinas utilizadas para execução do experimento
Fonte: Elaborada pelo graduando.

A máquina com o rótulo de *Necromancer* possui o IP 192.168.0.7 e será o nó distribuidor das informações. O código que será executado nessa máquina é ilustrado na figura 4.7.

```
Necromancer necromancer = Necromancer.getInstance();
necromancer.recruit(true);

ISomaSimples remoteSimpleMath = necromancer.undeadInstance( SomaSimples.class );
System.out.printf("soma: %d\n", remoteSimpleMath.soma(40, 2));
```

Figura 4.7: Código executado no *Necromancer*.
Fonte: Autor.

Na primeira linha, a instância única do *Necromancer* no processo é retornada, ou, se não existir, é criada. Na segunda linha, o modo de recrutamento é ativado, o qual permite aos clientes Zumbis conectarem-se ao *Necromancer*. Ressalta-se que antes da execução da segunda linha, o sistema ainda não responde as requisições de conexão. Na quarta linha, um objeto distribuído é criado, e, na linha seguinte, o método soma é chamado, passando os parâmetros 40 (quarenta) e 2 (dois). O resultado 42 (quarenta e dois) será, então, impresso na saída padrão.

A máquina utilizada para efetuar o cálculo será escolhida baseada na fila em que é criada, conforme as conexões dos zumbis cheguem.

A figura 4.8 contém o diagrama de fluxo da operação.

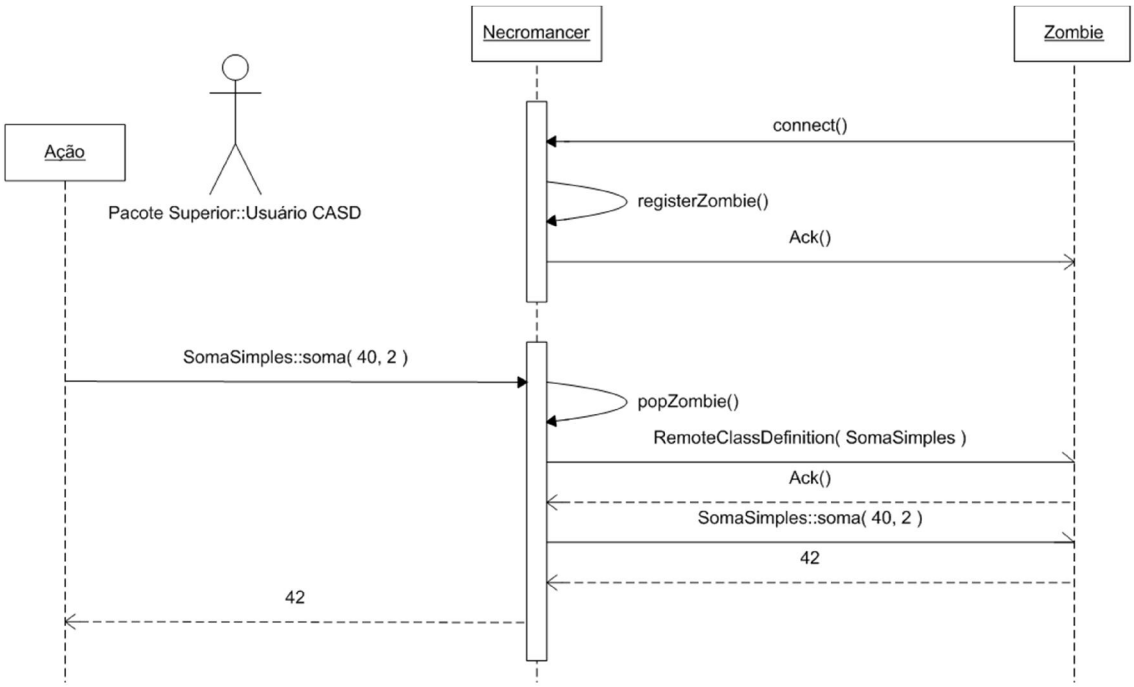

Figura 4.8: Diagrama de fluxo da operação.
Fonte: Autor.

A figura 4.8 reflete todo o fluxo computacional que é executado no programa. Na coluna esquerda, é possível perceber que o usuário executa a operação e obtém o retorno sem ter conhecimento do que ocorre dentro da biblioteca, como a obtenção de um zumbi, o envio da definição binária da classe ou a troca de informações de confirmação. A figura 4.7 ilustra como é a execução do ponto de vista do programador e é complementada pela figura 4.8.

A figura 4.9 abaixo reflete a implementação do método *Necromancer::undeadInstance*, que é o método *template* responsável por criar um *proxy* para o objeto que se deseja executar remotamente. O método recebe três parâmetros: *Zombie*, *Class*< T > e os parâmetros do construtor; o valor de retorno depende do tipo concreto da classe que se deseja instanciar.

```
public <T> T undeadInstance(Zombie z, Class<T> clazzType, Object...params )
{
        T infectedObject = null;
        try
        {
                infectedObject = (T) Proxy.newProxyInstance(
                                clazzType.getClassLoader(),
                                clazzType.getInterfaces(),
                                new RemoteClassProxy(this, z, clazzType, params));
        }
        catch (Exception ex)
        {
                Logger.getLogger(Necromancer.class.getName()).log(
                                Level.SEVERE, null, ex
                );
        }
        catch (Throwable t)
        {
                Logger.getLogger(Necromancer.class.getName()).log(
                                Level.SEVERE, null, t
                );
        }
        return infectedObject;
}
```

Figura 4.9: Implementação do método estático *Necromancer::undeadInstance*.
Fonte: Autor.

Dentro da implementação do método, existe a criação de uma nova instância da classe *RemoteClassProxy*, que é responsável por manter um estado interno, relacionando o zumbi alocado e a classe que deverá ser executada remotamente. Esse estado define se a definição binária da classe deve ou não ser enviada para o zumbi alocado para execução.

O método mais importante da classe *RemoteClassProxy* é o *invoke.* Esse método representa uma sobrecarga e é chamado sempre que qualquer método do objeto *proxy* for chamado. É nesse momento que a aplicação solicita que a execução do método seja realizada remotamente, conforme é mostrado na figura 4.10.

```
@Override
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable
{
        Object result = null;
        if (null == _alocatedZombie)
        {
                throw new NullPointerException(
                        "Não há nenhum zumbi alocado para este objeto remoto"
                );
        }
        result = _necromantic.syncSend(_alocatedZombie,
                        new RemoteMethodCall(_proxyClass,
                        m.getName(),
                        m.getParameterTypes()).addParam(args));

        return ((Result)result).value();
}
```

Figura 4.10: Implementação do método *RemoteClassProxy::invoke.*
Fonte: Autor.

Conforme é ilustrado na figura 4.10, o método recebe três parâmetros: o objeto *proxy,* o método que foi chamado e os parâmetros que foram passados para o método. Na chamada desse método, a biblioteca assume que já existe um zumbi alocado para essa operação. Caso não haja uma exceção, a chamada é lançada para a camada superior.

Na linha em que o método *Necromancer::syncSend* é chamado, uma requisição para o zumbi é enviada, a qual trava a execução do programa até que o computador remoto responda a requisição. O envio dessa ocorre por meio do *Necromancer*, visto que detém o contexto de um zumbi; esse contexto contêm todos os canais de comunicação, de entrada e de saída, e uma *thread* responsável por receber os dados oriundos do zumbi concorrentemente.

Desta forma, a figura 4.11 ilustra a implementação do método *ZombieContext::sendCommand,* responsável pelo envio da requisição e bloqueio da execução da *thread* até que a resposta seja recebida.

```
public IMessage sendCommand(IMessage msg) throws InterruptedException, IOException
{
    final Object MUTEXT_REQUEST = new Object();
    final Envelope envelope = new Envelope();

    DataListener syncDataListener = new DataListener()
    {
        @Override
        public Object dataArraived(Zombie z, Data d)
        {
            synchronized( MUTEXT_REQUEST )
            {
                envelope.setZombie(z);
                envelope.setData(d);
                MUTEXT_REQUEST.notifyAll();
                return null;
            }

        }
    };


    _zombie.addDataListener( syncDataListener );
    sendASyncCommand(msg);
    synchronized( MUTEXT_REQUEST )
    {
        MUTEXT_REQUEST.wait();
    }

    _zombie.removeDataListener( syncDataListener );
    return envelope.getData();
}
```

Figura 4.11: Implementação do método *ZombieContext::sendCommand.*
Fonte: Autor.

Nas primeiras duas linhas do método, dois objetos são criados: *MUTEX_REQUEST* e envelope. O primeiro serve como semáforo para sincronizar as duas *threads* que irão concorrer com esse trecho de código, e o segundo é utilizado como área de memória compartilhada entre as duas *threads*.

Na sequência, um objeto anônimo do tipo *DataListener* é criado com o nome de *syncDataListener*. Esse objeto é um ouvinte temporário ao evento de mensagem remota e será desregistrado no momento que a resposta esperada chegar.

Ao final da implementação do método, o objeto armazenado dentro da variável envelope é retornado. Esse objeto é exatamente a resposta da execução remota do método.

Desta forma a biblioteca consegue fornecer ao programador um acesso transparente ao processamento distribuído, atingindo o objetivo maior da biblioteca CASD, que era garantir a transparência da aplicação.

A biblioteca CASD possui um total de 38 Classes, das quais as principais são expostas na figura 4.12, que representa o diagrama de classe simplificado do pacote *engine*.

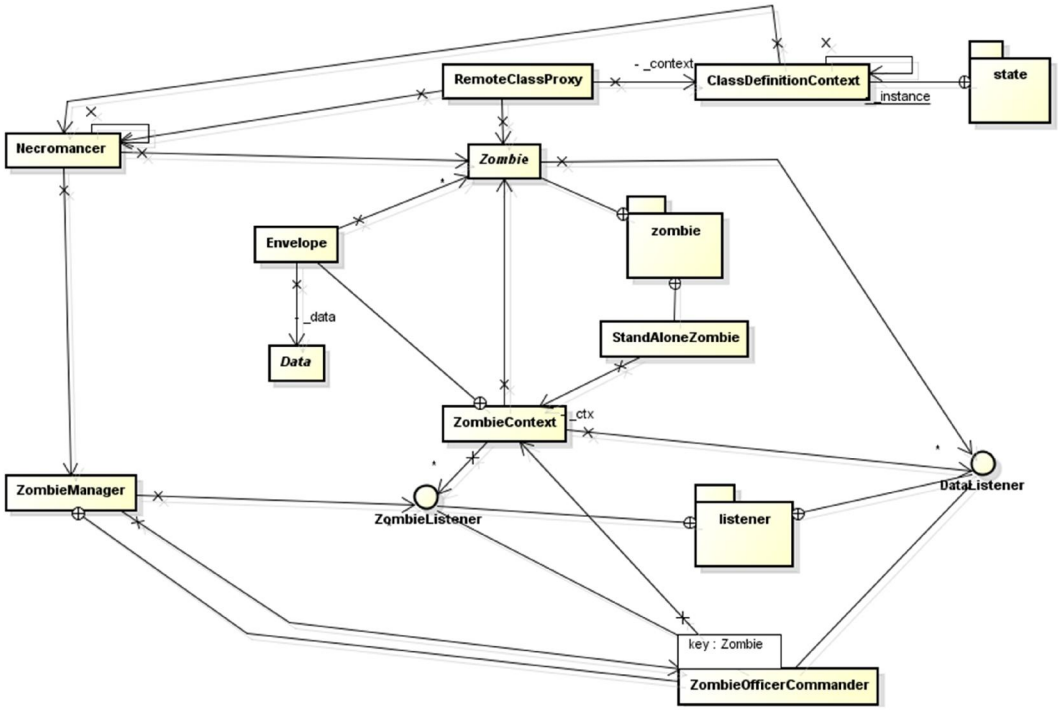

Figura 4.12: Diagrama de classe simplificado do pacote *engine.*
Fonte: Autor.

A figura 4.12 mostra a relação entre as classes e pacotes. Como a biblioteca contêm muitas classes, optou-se por exibir apenas as principais e seus relacionamentos. Os nós que se assemelham a uma pasta representam pacotes da linguagem de programação Java; os retângulos representam classes – abstratas ou concretas; e os círculos representam as interfaces. A ponta da seta fica sempre do lado da classe que contêm a referência, assim é possível visualizar as posses dos objetos formalmente.

Antes do início do desenvolvimento da biblioteca, foi executada uma etapa de projeto, na qual a biblioteca foi modelada e projetada, com base nos objetivos que eram almejados. O objetivo principal da biblioteca, como já mencionado, era oferecer o máximo de transparência para o programador no uso da aplicação e foi atingido.

A seguir, apresenta-se a outra biblioteca que compõe a ADD, a *JaxFish.*

4.1.2. Biblioteca "*JaxFish*"

A biblioteca *JaxFish* é especializada em realizar assinatura digital em arquivos XML e PDF, utilizando o certificado digital do usuário. O objetivo desta biblioteca foi atingido, visto que ela garante que o fluxo do código permanece constante mesmo se o tipo de certificado for modificado, independentemente se o algoritmo mudar ou se o dispositivo for diferente.

Como mencionado no Capítulo 1, o escopo deste trabalho ficou limitado à assinatura de arquivos PDF; por tanto, não será abordada a assinatura de arquivos XML.

Para utilização da biblioteca, é necessário possuir um certificado digital do tipo A1. O certificado utilizado neste projeto contêm as exatas informações que aparecem na figura 4.13.

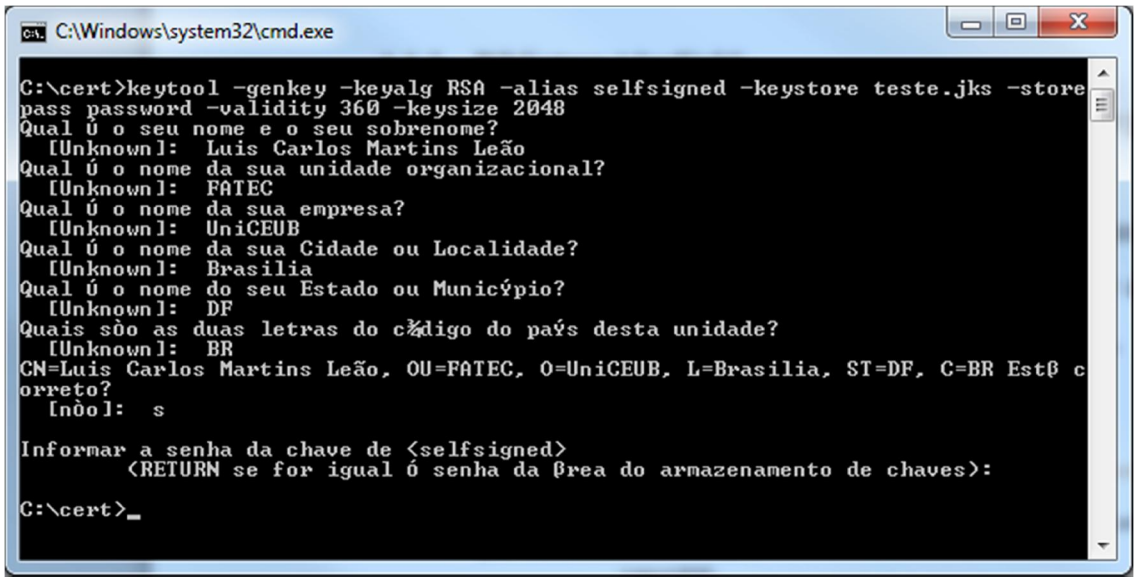

Figura 4.13: Informações contidas no certificado digital utilizado no projeto.
Fonte: Autor.

Inicialmente, criar-se uma interface que faça o acesso ao certificado de maneira homogênea. Na sequência, é necessário possuir um arquivo PDF para ser assinado. No exemplo apresentado na figura 4.14, o arquivo PDF a ser assinado chama-se casd.pdf e está localizado fisicamente na pasta C:\pdf.

```
DigitalCertificateInterface cert =
            new FileCertificateDevice("C:\\cert\\keystore.jks", "123456".toCharArray());
File inputPDF = new File("C:\\pdf\\casd.pdf");
AbstractSignablePDF pdf = FactorySignablePDF.getSignablePDF(cert);
pdf.assinarDocumentoPDF(
            inputPDF,
            new FileOutputStream( new File( inputPDF.getParent(), "assinado_" + inputPDF.getName() ) ),
            "Assinando para testar o assinador JaxFish",
            "UniCEUB");
```

Figura 4.14: Exemplo de documento PDF a ser assinado.
Fonte: Autor.

Na linha 1, a interface para acesso ao certificado é criada. Na linha 2, por sua vez, um objeto que representa o arquivo PDF é criado e passado como parâmetro à *AbstractSignablePDF::assinarDocumentoPDF* . Já na linha 3, uma fábrica cria o assinador abstrato de PDF, com base nas informações do certificado, como o tamanho da chave.

Após a execução dos procedimentos, um novo arquivo com o nome de *assinado_casd.pdf* será gerado na pasta C:\pdf. Este arquivo conterá a assinatura, utilizando o certificado, e garantirá a integridade do arquivo PDF.

Antes do desenvolvimento da biblioteca, na fase de projeto, foi levantado o que os certificados digitais possuem em comum, como chave privada, chave pública, certificado do usuário e a cadeia de certificados. De posse dessas informações, foi criada a interface chamada de *DigitalCertificateInterface,* a qual expõe essas informações conforme a figura 4.15.

```
public interface DigitalCertificateInterface
{
    public PrivateKey getPrivateKey();
    public PublicKey getPublicKey();
    public X509Certificate getUserCertificate()throws KeyStoreException;
    public X509Certificate[] getCertificateChain() throws KeyStoreException;

}
```

Figura 4.15: Interface que expõe os métodos comuns dos certificados.
Fonte: Autor.

A interface possui duas implementações concretas, porém apenas uma delas será abordada nesse tópico, a *FileCertificateDevice.* Essa implementação é especializada em ler um certificado do tipo A1, o qual pode estar localizado na máquina ou em algum outro computador, desde que haja um caminho válido para a criação de uma *InputStream*.

De posse do certificado que será utilizado para assinar o documento PDF, cria-se o assinador. Para tal função, existe a classe fábrica *FactorySignablePDF*, a qual é utilizada para selecionar o algoritmo que será utilizado na assinatura do documento, com base no certificado digital passado como parâmetro para o método estático *FactorySignablePDF:: getSignablePDF(DigitalCertificateInterface)*. O retorno desse método é um objeto do tipo *AbstractSignablePDF*. Esse objeto é especializado em assinar arquivos PDF e é o principal objeto da biblioteca JaxFish.

Na sequência, passa-se à descrição dos passos necessários para desenvolvimento do modelo de Assinatura Digital Distribuída.

4.2. Descrição das Etapas para o desenvolvimento da ADD

Neste subitem, é apresentado o passo-a-passo do desenvolvimento do modelo proposto neste trabalho, a Assinatura Digital Distribuída. Inicia-se com as etapas do desenvolvimento da CASD e da *JaxFish* e, na sequência, descreve-se como estas foram utilizadas dentro da ADD. As etapas aqui apresentadas não derivam de metodologias específicas, mas foram elaboradas pelo autor do projeto.

4.2.1. Desenvolvimento da CASD

Abaixo são apresentadas as etapas sequenciais do desenvolvimento da biblioteca CASD:

a. Criação de um componente servidor e um componente cliente:
    i. Criação de um *ServerSocket* com utilização da biblioteca *java.net* que ouve as requisições na porta 8734, a qual foi definida arbitrariamente;
    ii. Criação de um *Socket* utilizando a biblioteca *java.net* que se conecta no computador local via o IP 127.0.0.1 na porta 8734.
b. Conexão entre os componentes:

   i. Solicitar que o componente cliente conecte-se ao componente servidor para estabelecer o canal de comunicação em cima de TCP/IP;
   ii. Envio de dados para teste partindo de ambos os lados.

c. Envio da definição binária de uma classe para o cliente:
   i. Criação de uma classe de teste para ser enviada ao cliente;
   ii. Descobrir a localização física deste binário, utilizando o *ClassLoader* do *Java,* e enviar o arquivo localizado para o cliente;
   iii. Descobrir as dependências da Classe, considerando outras classes, interfaces e *annotations*.

d. Execução da classe binária:
   i. Depois de receber os dados do servidor, salvar a classe como arquivo em um diretório relativo à execução da aplicação, respeitando a hierarquia de pastas baseado no *package* da classe;
   ii. Executar arbitrariamente algum método estático que exista na classe;
   iii. Executar o método solicitado pelo servidor;
   iv. Criação de um encapsulamento para o envio da definição da classe e do método deve ser executado;
   v. Adicionar ao encapsulamento a possibilidade de passar parâmetros.

e. No servidor, utilizar o padrão de projetos *Proxy* para a criação de um objeto remoto:
   i. Para a inicialização remota de um objeto é necessário utilizar uma rotina diferente do operador *new* e chamar o método estático do *Necromancer undeadInstance*, o que irá criar um *proxy* para capturar todas as chamadas a métodos. Ao invés de executá-los localmente, o *proxy* submete as requisições ao cliente remoto;
   ii. Um identificador é criado para cada objeto; assim, enquanto objeto existir no escopo do *Necromancer*, ele deverá existir no cliente.

f. Execução de um método específico de um objeto no cliente, recebendo o resultado da execução:
   i. Supondo um método que retorne um inteiro que é a soma dos dois parâmetros do método, enviar o resultado para o servidor.

4.2.2 Desenvolvimento da *JaxFish*

As etapas necessárias para o desenvolvimento da biblioteca *JaxFish* são listadas a seguir:

a. Estudo das RFC's 3280 e 5280, do documento PDF *Reference* 1.7 e da biblioteca *Bouncy Castle*.
b. Engenharia reversa em um assinador de XML que já utilizava assinatura digital.
    i. A engenharia reversa aplicada neste assinador teve como intuito entender o funcionamento do fluxo de uma assinatura digital e a utilização dos algoritmos;
    ii. Para a engenharia reversa, foi utilizado o decompilador presente na ferramenta de depuração do Java, que já vem por padrão instalada junto com a JDK do Java. O decompilador gera um código semelhante ao *assembly*, o qual foi analisado para compreender o funcionamento do assinador.
c. Criação de um modelo de acesso abstrato ao certificado.
    i. Como a assinatura pode ser realizada em qualquer tipo de certificado, foi criada uma camada abstrata que expõe as ações comuns de qualquer certificado. Assim, o sistema pode funcionar de maneira homogênea, independentemente da implementação adotada;
    ii. Existem dois usos mais comuns: o certificado em arquivo (A1) e o certificado em *token* (A3).
d. Criação de um componente de leitura de PDF.
    i. Após o estudo do documento PDF *Reference* 1.7, foi possível criar um componente que lê o arquivo PDF e consegue calcular a assinatura digital desse arquivo.
e. Assinatura digital de um PDF, utilizando o componente de acesso abstrato ao certificado e aplicando as regras de assinatura condidas no PDF *Reference* 1.7.

i. Utilizando o componente criado no item “c”, juntamente com o componente do item “d”, é possível a realização da assinatura de um documento PDF.

4.2.3 Desenvolvimento da ADD

Após o desenvolvimento das bibliotecas CASD e *JaxFish*, estas foram combinadas para o desenvolvimento da ADD, cujas etapas são apresentadas abaixo:

b. Criação de uma classe que implementasse *Runnable*, que recebesse um certificado digital e assinasse um PDF, utilizando a biblioteca JaxFish;
c. Instanciação da classe criada no item “a”, utilizando a CASD para criar uma instância remota;
d. Execução do objeto remoto para assinar o PDF;
e. Criação de um *Pool* de *Threads*;
f. Submeter a execução do item “c” ao *Pool* de *Threads* para não bloquear a execução principal da ADD;
g. Obter o resultado da assinatura em *bytes* e salvá-lo em arquivo local com o mesmo nome do arquivo original, porém com o sufixo “assinado_”.
h. Verificar a assinatura do arquivo.

Após descritos os passos necessários para o desenvolvimento da ADD e das respectivas bibliotecas que a compõem, segue-se à descrição das etapas necessárias à implementação do modelo.

4.3. Descrição das Etapas necessárias para Implementação da ADD

A seguir, segue a descrição dos passos necessários para implementação prática do modelo desenvolvido neste projeto, a Assinatura Digital Distribuída:

a. Para aplicação da ADD, inicialmente, é necessário possuir um certificado digital do tipo A1. Caso não se deseje comprar um certificado e caso o documento não necessite possuir validade jurídica, é possível criar um certificado autoassinado;
b. Na sequência, deve-se instalar o ADD em uma máquina. Como a aplicação ADD foi escrita em Java, basta descompactá-la em qualquer diretório;
c. Após, faz-se necessário colocar todos os arquivos em uma pasta qualquer na máquina onde a ADD foi instalado, preferencialmente em alguma pasta local que seja acessível pelo *Necromancer*. Os zumbis irão acessar os arquivos de forma transparente, como se o documento estivesse na máquina cliente;
d. Deve-se, então, iniciar a aplicação ADD, executando em uma console o comando *java –jar add-casd.jar*;
e. A seguir, necessita-se configurar a ADD para acessar a pasta definida no item "c";
f. O próximo passo é configurar a ADD para acessar o diretório aonde os arquivos PDF's assinados devem ser salvos;
g. Também é necessário configurar na aplicação a localização do certificado digital A1;
h. Nas máquinas que serão utilizadas como zumbi, deve-se salvar a aplicação *DummyZombie.jar* e executar o comando em uma console *java –jar DummyZombie.jar* para que a estação esteja pronta para ceder seus recursos ao *Necromancer*;
i. A seguir, é necessário clicar no botão INICIAR na aplicação ADD. Esta será executada apenas na máquina que será o *Necromancer* e que irá coordenar a forma com que os procedimentos serão executados;
j. O usuário deve, então, fornecer a senha de acesso ao certificado digital quando for solicitado;
k. Por fim, deve-se aguardar a finalização do procedimento de assinatura;
l. Depois de terminada a assinatura de todos os arquivos, um procedimento de verificação manual deve ser executado.

CAPÍTULO 5 - APLICAÇÃO PRÁTICA DA ASSINATURA DIGITAL DISTRIBUÍDA (ADD)

A aplicação prática, envolvendo um caso real de aplicação, tem como objetivo principal demonstrar a viabilidade da proposta de resolução sugerida no projeto, bem como permitir que novos conhecimentos sejam incorporados.

Dessa forma, este Capítulo apresenta as informações da aplicação prática da Assinatura Digital Distribuída (ADD), iniciando-se com uma breve descrição da área em que o modelo foi implementado. Na sequência, descrevem-se os passos para aplicação do projeto e expõem-se os resultados atingidos e os custos envolvidos. Por fim, apresenta-se uma avaliação global do modelo.

5.1. Apresentação da área de Aplicação da ADD

A ADD foi testada na empresa Sigma *Dataserv* S.A., a qual presta serviços terceirizados de consultoria em informática para o Ministério do Desenvolvimento e Comércio Exterior (MDIC) em diversas áreas, as quais incluem auditoria em sistemas, integração de sistemas, planejamento estratégico em tecnologia da informação, modelagem de soluções, *software* livre e seleção de fornecedores e tecnologias. A Sigma *Dataserv* também desenvolve soluções em *e-business* e trabalha com o *outsourcing* de aplicações (SIGMA, 2013).

Mais especificamente, os testes foram realizados na Coordenação Geral de Modernização e Informática (CGMI) da referida empresa, departamento responsável por gerir as coordenações de Desenvolvimento e Infraestrutura do MDIC e prover a sinergia entre essas, administrando os contratos para as demandas intra e extraministerial.

Em comprimento à Medida Provisória (MP) nº. 2.200-2/2001 e à Lei nº. 11.419/2006, a empresa decidiu assinar digitalmente 15.000 (quinze mil) documentos PDF de caráter reservado. Por não terem caráter público, a natureza e o conteúdo dos arquivos não serão abordados.

Na sequência, apresentam-se os detalhes sobre os testes de aplicação da ADD realizados na Sigma *Dataserv* S.A.

5.2. Descrição da Aplicação da ADD

O testes de aplicação da ADD na supracitada empresa seguiram os seguintes estágios:

a. Criação do certificado A1 de máquina. Foi necessária a aquisição do certificado digital para que os documentos possuíssem validade jurídica, em cumprimento à MP nº. 2.200-2/2001 e à Lei nº. 11.419/2006. Para atingir o objetivo, os seguintes procedimentos foram executados consecutivamente:
    i. O comando *keytool -certreq -file teste.crq* foi executado para criar um arquivo contendo as informações necessárias para se obter um certificado A1 válido, pertencente a hierarquia da ICP-Brasil;
    ii. O arquivo teste.crq foi aberto com o editor de texto *Notepad++* e seu conteúdo foi copiado para a área de trabalho, o qual detém a informação da chave pública gerada pela ferramenta *keytool.* Fez-se necessário, então, enviar essa informação para o SERPRO para que ele assinasse digitalmente também a chave da empresa e tornasse o certificado válido para fins jurídicos;
    iii. Foi enviado para o SERPRO o conteúdo da área de trabalho pela página https://ccd.serpro.gov.br/ACPR/. Selecionou-se o menu à esquerda "Solicitar Certificado A1 – Equipamento" e colou-se o conteúdo no campo "Dados da solicitação do certificado";
    iv. Os dados requeridos na página foram preenchidos. Aqui os dados não serão expostos por se trataram de dados particulares da instituição.
b. Todos os documentos foram agrupados em uma única máquina de IP 10.27.18.39:
    i. O aplicativo ADD foi copiado para a área de trabalho na pasta ADD;

ii. O comando de execução no DOS (*Windows*) ou no terminal (*Linux*) utilizado foi *java –jar ADD.jar*. Então, a aplicação foi colocada em execução;
iii. Criou-se a pasta D:\arquivos_importacao nessa máquina;
iv. Criou-se também a pasta D:\arquivos_assinados nessa máquina;
v. Todos os arquivos PDF que deveriam ser assinados foram colocados dentro da pasta citada no item iii.

c. A entidade Zumbi foi instalada em 15 máquinas, fora a máquina descrita no item b:
    i. O aplicativo *DummyZombie*.jar foi copiado para a área de trabalho de todas as máquinas em uma pasta chamada *DummyZombie*;
    ii. O comando de execução no DOS (*Windows*) ou no terminal (*Linux*) utilizado foi *java –jar DummyZombie*.jar;
    iii. No primeiro teste, o comando do item ii foi executado em apenas 5 máquinas, contendo mil arquivo PDF a ser assinados;
    iv. Nos testes seguintes, foram sendo adicionadas 5 máquinas por vez, até atingir a quantidade de 15 máquinas executando o procedimento, com o intuito de medir a variação no tempo da execução.

d. Execução do procedimento:
    i. Na máquina em que o ADD estava em execução, as pastas origem e destino foram configuradas com os diretórios descritos nos itens b.iii e b.iv;
    ii. Conforme descrito no item c.iv, o procedimento foi executado 3 vezes para teste, sempre adicionando novas máquinas à rede de processamento. A execução do procedimento se deu clicando no botão “Processar”;
    iii. Por fim, foi executado o procedimento para assinatura de 15.000 arquivos PDF, utilizando as máquinas.

e. Após cada execução, avaliou-se por meio de validação manual por amostragem se os arquivos estavam sendo corretamente assinados. A amostragem foi feita de forma arbitrária e subjetiva. A cada 200 arquivos, uma assinatura foi verificada.

5.3. Resultados da Aplicação da ADD

Em teoria, cada arquivo PDF leva, em média, 2,5 segundos para ser assinado. Dessa forma, o tempo total para assinar sequencialmente os 15.000 documentos PDF da empresa Sigma *Dataserv* S.A. que compuseram o escopo dos testes de aplicação deste projeto seria de aproximadamente 10 horas e 30 minutos. Ao paralelizarmos o processamento usando a CASD em 15 máquinas, teríamos um tempo de espera teórico de 40 minutos.

Conforme exposto no item 5.2, o teste inicial da ADD foi realizado em cinco máquinas com 1.000 arquivos. O resultado obtido foi de um tempo de assinatura de 9,75 minutos, superior ao tempo teórico esperado para esses parâmetros, que era de 8,4 minutos.

Na sequência, adicionaram-se mais cinco máquinas ao sistema, chegando ao total de 10 zumbis, porém mantendo os mesmos 1.000 arquivos. O tempo requerido para todas as assinaturas foi de 5,47 minutos, contra o resultado teórico de 4,17 minutos.

Por fim, adicionaram-se mais cinco máquinas ao sistema, totalizando 15 máquinas zumbis. O tempo resultante da assinatura dos 1.000 arquivos foi de 3,47 minutos, tendo como tempo teórico 2,7 minutos.

Observou-se diminuição gradativa do tempo requerido para assinar os 1.000 documentos, conforme novas máquinas foram inseridas no sistema. Logo, infere-se que se mais máquinas forem utilizadas para a assinatura dessa mesma quantidade de arquivos, o tempo tende a diminuir ainda mais.

A figura 5.1. a seguir ilustra os resultados obtidos pela execução.

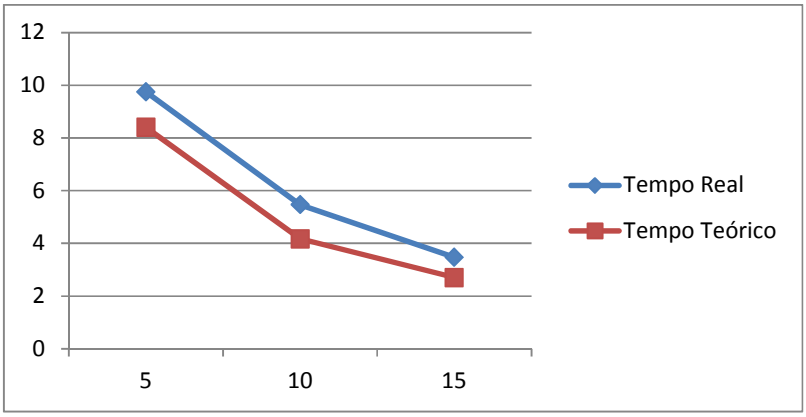

Figura 5.1: Resultado obtido com os testes com 1.000 arquivos.
Fonte: Autor.

A linha azul expressa o tempo real que a aplicação levou para executar o procedimento completo e a linha vermelha, o tempo teórico, que desconsidera qualquer fator não pertencente à assinatura digital do documento. É perceptível que, conforme mais máquinas são adicionadas à grade computacional, menor é o tempo de processamento. No entanto, esse tempo tende a convergir, pois haverá um momento que a rede irá saturar devido à quantidade de máquinas compartilhando os recursos.

O gráfico da figura 5.1 apresenta um coeficiente angular negativo, ou seja, uma curva decrescente, o que se coaduna com o objetivo do projeto de tornar o processo mais rápido, conforme novas máquinas forem adicionadas à grade computacional.

Como os testes com 1.000 arquivos foram bem sucedidos, passou-se à assinatura de todos os 15.000 documentos PDF da empresa. Esses arquivos foram assinados em 3.250 segundos, o que equivale a aproximadamente 55 minutos.

Os 15 minutos que aparecem entre o cálculo esperado – apresentado no início desse subitem – e o resultado encontrado devem-se à infraestrutura de rede – que, por mais rápida e eficiente que seja, acaba gerando uma latência na comunicação entre os dispositivos – e ao tempo requerido para ler e escrever um arquivo no disco rígido do computador.

Destaca-se, também, que a execução dos procedimentos foi transparente, uma vez que não foi possível perceber que outras máquinas compunham a rede, nem quando eram adicionadas ou removidas, impactando diretamente no tempo de processamento total dos arquivos.

Ademais, a máquina que executava a aplicação ADD não ficou indisponível em nenhum momento, alcançando-se, assim, um dos objetivos principais da aplicação, que era não tornar o recurso indisponível para outras operações.

5.4. Custos da ADD

A seguir, são listados os custos envolvidos nos testes de aplicação da ADD:

a. Aquisição de um certificado digital do tipo A1:
    i. Como a empresa em que foram realizados os testes não possuía certificado digital do tipo A1 e necessitava que os documentos

assinados possuíssem validade jurídica, foi necessária a aquisição do referido certificado no SERPRO, totalizando R$764,80 (setecentos e sessenta e quatro reais e oitenta centavos).

b. Mão-de-obra (horas de desenvolvimento):
    i. A aplicação alocou o tempo de aproximadamente 170 horas;
    ii. O valor estimado da hora foi de R$73,13 (setenta e três reais e treze centavos).
    iii. Logo, multiplicando-se o tempo total e o valor por hora, teve-se como valor final de custo de mão-de-obra de R$12.431,25 (doze mil quatrocentos e trinta e um reais e vinte e cinco centavos).

b. Aquisição de equipamentos:
    i. Não foi necessária a aquisição de um servidor com alto poder de processamento, visto que a CASD viabiliza que as máquinas já existentes no parque tecnológico da empresa sejam utilizadas em conjunto, permitindo a paralelização das assinaturas dos PDF's. Logo, o poder de processamento das máquinas é somado.

c. Custos de ociosidade:
    i. Os testes da ADD não incorreram em custos de ociosidade, uma vez que não foi necessário deslocar os funcionários de suas atividades, já que o procedimento foi executado em horário fora do expediente normal de trabalho, não interrompendo, assim, nenhuma atividade regular da empresa.

5.5. Avaliação Global da ADD

Em uma avaliação geral, a aplicação garante uma rápida execução de tarefas, que antes eram executadas sequencialmente em um único computador, em vários computadores de forma paralela e concorrente, multiplicando, assim, o poder de processamento do sistema.

Sabe-se que um dos maiores desafios para a programação paralela é treinar o desenvolvedor para gerir os recursos e controlar o ciclo de vida de um objeto. Nesse sentido, a utilização da CASD representa uma vantagem, uma vez que essas complexidades são delegadas à biblioteca.

Outras vantagens da aplicação estão relacionadas aos baixos custos que tiveram de ser despendidos pela empresa e à possibilidade de as bibliotecas CASD e *JaxFish* serem reutilizadas para outras finalidades. A CASD pode ser incorporada a quase qualquer tipo de processamento paralelo. A *JaxFish*, por sua vez, pode ser utilizada para assinar documentos PDF e XML, utilizando certificados A1 ou A3, e pode ser usada como autenticador de usuário por meio de *token*.

A ADD também apresenta vantagens. Como foi criada para atender a demanda de assinar arquivos PDF ligados, os novos arquivos que forem gerados serão assinados automaticamente pela *JaxFish*, não sendo mais necessário assinar uma grande quantidade de documentos.

Diante do exposto, verifica-se que os benefícios gerados pelas bibliotecas isoladamente e pela ADD a curto e longo prazo são satisfatórios.

No entanto, encontrou-se um problema na aplicação. A biblioteca CASD detecta os zumbis por meio de um *broadcast* na rede, porém Java contém um *bug* no momento de se obter o endereço de *broadcast* da rede. Este problema ocorre apenas em algumas redes sem fio e não pode ser solucionado, pois o *bug* na versão 7 do Java ainda não havia sido corrigido no momento dos testes da ADD. De forma paliativa, adicionou-se à biblioteca a possibilidade de se configurar o endereço de *broadcast* via linha de comando.

CAPÍTULO 6 – CONSIDERAÇÕES FINAIS

Este capítulo final apresenta as conclusões deste projeto, evidenciando se o problema e os objetivos do trabalho foram alcançados, bem como se os resultados atingidos com a aplicação do modelo foram satisfatórios. Também são abordadas limitações do projeto e sugestões para trabalhos futuros.

6.1. Conclusões

Verifica-se que as ferramentas disponíveis no mercado para assinar documentos digitalmente são muito caras e executam a assinatura em lote em uma única máquina. Portanto, quando se necessita assinar um volume grande de documentos, é possível levar dias para concluir a tarefa. Outro efeito colateral além da demora é a indisponibilidade da máquina utilizada para assinar digitalmente os documentos.

Ponderando sobre essa questão, pode-se retomar o problema de pesquisa proposto nesse trabalho: Como assinar digitalmente documentos de maneira ágil e econômica?

Com os constantes avanços tecnológicos, percebe-se que existe a tendência de convergir a computação local para computação distribuída ou computação nas nuvens, de forma as aplicações mais ágeis e a diminuir os custos de processamento.

Nesse sentido, a biblioteca CASD oferece as funcionalidades necessárias para utilizar os recursos computacionais de outras máquinas de forma distribuída, de maneira transparente e escalável, o que representa uma alternativa às ferramentas que usam apenas os recursos locais para assinar documentos digitalmente. A biblioteca *JaxFish*, por sua vez, abstrai o acesso aos diferentes tipos de certificados digitais do usuário, o que garante a transparência da aplicação e gera flexibilidade ao uso da assinatura.

Conclui-se que o problema pode ser solucionado após juntar as bibliotecas CASD e *JaxFish* para dar origem ao sistema ADD. A *JaxFish* efetua a assinatura de documentos PDF e a CASD consegue executar qualquer objeto de forma remota. Logo, atingiu-se o objetivo geral desse projeto, que era o de especificar, desenvolver e implantar sistema que permita

assinar uma quantidade arbitrária de documentos PDF, utilizando computação distribuída, de tal forma que não seja possível determinar de qual máquina os recursos foram utilizados.

Ao alcançar o Objetivo Geral, validam-se também os objetivos específicos apresentados pelo estudo:

a) Dissertar sobre computação distribuída e assinatura digital;
b) Criar um software capaz de suprir o problema;
c) Testar a aplicação; e
d) Avaliar os resultados atingidos.

O objetivo específico (a) foi alcançado na apresentação dos tópicos 2.2 e 2.3, intitulados “Sistemas Computacionais Distribuídos” e “Assinatura Digital”, respectivamente. O objetivo específico (b) foi abordado no Capítulo 4, o qual descreve todo o desenvolvimento das bibliotecas CASD e *JaxFish*. Por fim, os objetivos (c) e (d) foram evidenciados no Capítulo 5, nos itens 5.2, 5.3 e 5.5.

6.2. Sugestões para Trabalhos Futuros

Visto que um trabalho de pesquisa nunca se esgota em si, abaixo são levantados tópicos complementares que viabilizariam pesquisas futuras em continuidade aos resultados alcançados com este projeto.

a) Implementar um algoritmo de escalonamento para seleção de um zumbi baseado na complexidade da operação e no *delay* entre o *Necromancer* e o zumbi;
b) Utilizar conexões seguras, como SSL, para a comunicação entre o *Necromancer* e os Zumbis.

REFERÊNCIAS

ADOBE. *PDF Reference Sixth Edition: Adobe Portable Document Format Version 1.7*. 2006. Disponível em: < http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_ reference_1-7.pdf>. Acesso em: 11 jun 2013.

BRAGA, Ataualpa A. Carmo. Aspectos técnicos envolvidos na construção de um "cluster beowulf". *Química Nova*, Campinas, v. 26, n. 3, p. 401-406, 2003.

BRASIL. *Medida Provisória nº. 2002-2, de 24 de agosto de 2001*. Institui a Infraestrutura de Chaves Públicas Brasileiras – ICP-Brasil, transforma o Instituto Nacional de Tecnologia em autarquia, e dá outras providências. Brasília, 2001. Disponível em: <https://www.planalto.gov.br/ccivil_03/MPV/Antigas_2001/2200-2.htm>. Acesso em: 15 jun 2013.

BOUNCY Castle. *[Definição]*. 2013. Disponível em: < http://www.bouncycastle.org/ index. html>. Acesso em: 5 jun 2013.

COMPUTAÇÃONUVEM. *Modelos de serviço*. 2012. Disponível em: <http://computacaonuvem.wordpress.com/modelos-de-servico/>. Acesso em: 30 maio 2013.

CONTI, Fabieli de. *Grades computacionais para processamento de alto desempenho.* Santa Maria, 2009. Disponível em: <http://www-usr.inf.ufsm.br/~andrea/elc888/artigos/artigo3. pdf>. Acesso em: 30 maio 2013.

COOPER, D.; SANTESSON, S.; FARRELL, S.; BOEYEN, S.; HOUSLEY, R.; POLK, W. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revogation List (CRL) Profile.* 2008. Disponível em: < http://www.ietf.org/rfc/rfc5280.txt>. Acesso em: 10 jun 2013.

CORREA, Miguel Pupo. *Sociedade de Informação e Direito: a Assinatura Digital.*1999. Disponível em: < http://www.egov.ufsc.br/portal/sites/default/files/anexos/13816-13817-1-PB.htm#13>. Acesso em: 19 jun 2013.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. *Sistemas Distribuídos: Conceitos e Projeto*. 4. ed. São Paulo: Bookman, 2007.

D'ÁVILA, César Kyn. *RFC – Request for Comments*. 2009. Disponível em: <http://www.cedet.com.br/index.php?/O-que-e/Internet-e-Convergencia/rfc-request-for-comments.html>. Acesso em 7 jun 2013.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, Jonh. *Design Patters: elements of reusable object oriented software*. 1. ed. Massachusetts: Addison-Wesley, 2009.

IBM. *Introdução à programação JAVA, Parte 2: Desenvolvimentos para aplicativos reais*. Serialização do JAVA. 2011. Disponível em: < http://www.ibm.com/developerworks/br/java/ tutorials/j-introtojava2/section11.html>. Acesso em: 20 jun 2013.

JF. *O que é assinatura digital?* 2013. Disponível em: <http://www.jf.jus.br/cjf/tecnologia-da-informacao/identidade-digital/o-que-e-assinatura-digital>. Acesso em: 19 jun 2013.

KAZIENKO, Juliano Fontoura. *Assinatura digital de documento eletrônicos através da impressão digital*. 2003. 137 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação: Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2003.

LEITE, Jair C. *Engenharia de Software*. 2007. Disponível em: <http://engenhariadesoftware.blogspot.com.br/2007/02/sistemas-computacionais.html>. Acesso em: 5 jun 2013.

LOPES, Arthur Vargas. *Estrutura de dados para construção de software.* 2. ed. Canoas: Ulbra, 1999.

MAASEN, Jason; VAN NIEUWPOORT, Rob; VELDEMA, Ronald; BAL, Henri; KIELMANN, Thilo; JACOBS, Ceriel; HOFMAN, Rutger. Efficient Java RMI for Parallel Programming. *Journal ACM Transactions on Programming Languages and Systems*, 2001.

MICROSOFT. *Reflexão (C# e Visual Basic)*. 2013a. Disponível em: <http://msdn.microsoft.com/pt-br/library/vstudio/ms173183.aspx>. Acesso em: 20 jun 2013.

MICROSOFT. *Serialização (C# e Visual Basic)*. 2013b. Disponível em: <http://msdn.microsoft.com/pt-br/library/vstudio/ms233843.aspx>. Acesso em: 20 jun 2013.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. *Sistemas operacionais*. 3. ed. Porto Alegre: Sagra Luzzatto, 2008.

ORACLE. *Trail: the Reflection API*. 2013. Disponível em: < http://docs.oracle.com/javase/ tutorial/reflect/>. Acesso em: 20 jun 2013.

PAVAN, Willingthon. *Tolerância a falhas e Reflexão computacional num ambiente distribuído.* 2000. 86 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação: Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

PITANGA, Marcos. *Construindo supercomputadores com Linux*. 3. ed. Rio de Janeiro: Brasport, 2008.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Distributed Systems: Principles and Paradigms*. 2. ed. Amsterdã: Prentice Hall, 2006.

TANENBAUM, Andrew S. *Organização Estruturada de Computadores*. 5. ed. São Paulo: Prentice Hall, 2007.

TAURION, Cezar. *Cloud computing: computação em nuvem: transformando o mundo da tecnologia da informação*. 1ª ed. Rio de Janeiro: Brasport, 2009.

APÊNDICES

Apêndice A – Classes da Biblioteca CASD

1 Discover

1.1 NecromancerDiscover

2 Engine

2.1 Necromancer

2.2 ZombieContext

2.3 Ex

2.3.1 RemoteMethodCallException

2.3.2 UnreachableNecromancer

2.4 Listener

2.4.1 AckListener

2.4.2 ControlListener

2.4.3 DataListener

2.4.4 ExceptionListener

2.4.5 ZombieListener

2.5 Object(pool)

2.5.1 ObjectPool

2.6 State

2.6.1 ClassDefinitionContext

2.6.2 DefinitionState

2.6.3 DefinitionInitialState

2.6.3 DefinitionSynchronizedState

2.7 Visitor

2.7.1 AckMessageVisitor

Apêndice B – Classes da Biblioteca *JaxFish*

1 certificate

1.1 AbstractDigitalCertificate

1.2 DigitalCertificateInterface

2 http

2.1 HttpPost

2.2 PostClient

3 keymanager

3.1 CustomKeyManager

4 token.loader

4.1 FactorySignablePDF

4.2 FileCertificateDevice

4.3 LoadTokenDevice

5 token.model

5.1 AbstractSignablePDF

6 token.model.algs

6.1 Sha1WithRSASignableXML

6.2 Sha256WithRSASignableXML

7 token.model.event

7.1 SignEvent

7.2 SignedListener

8 token.loader.ui

8.1 PasswordDialog

9 trustmanager

9.1 CustomTrustManager

Apêndice C – Código Fonte da Biblioteca CASD

Este apêndice encontra-se no CD anexo a esse projeto.

Apêndice D – Código Fonte da Biblioteca *JaxFish*

Este apêndice encontra-se no CD anexo a esse projeto.