



Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Carolina Yoshico Ji

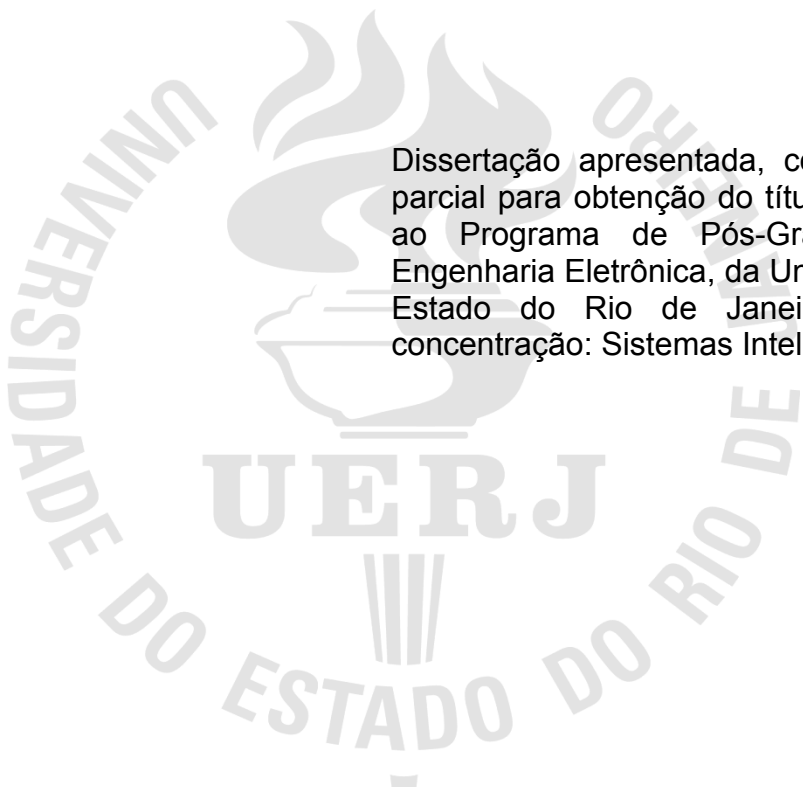
**Um Sistema Nebuloso de Detecção de Intrusos para Computação
em Nuvem**

Rio de Janeiro

2013

Carolina Yoshico Ji

Um Sistema Nebuloso de Detecção de Intrusos para Computação em Nuvem



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes

Orientador: Prof. Dr. Nival Nunes de Almeida

Orientador: Prof. Dr. Orlando Bernardo Filho

Rio de Janeiro

2013

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

J61	<p>Ji, Carolina Yoshico. Um sistema nebuloso de detecção de intrusos para computação em nuvem / Carolina Yoshico Ji - 2013. 231f. : il.</p> <p>Orientador: Nival Nunes de Almeida Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.</p> <p>1. Engenharia Eletrônica. 2. Lógica nebulosa - Dissertações. 3. Computação em nuvem - Dissertações. I. Almeida, Nival Nunes. II. Universidade do Estado do Rio de Janeiro. III. Título.</p> <p>CDU 004.89</p>
-----	--

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta dissertação, desde que citada a fonte.

Assinatura

Data

Carolina Yoshico Ji

Um sistema nebuloso de detecção de intrusos para computação em nuvem

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas inteligentes.

Aprovada em: 16 de agosto de 2013.

Banca examinadora:

Prof. Dr. Nival Nunes de Almeida (Orientador)
Faculdade de Engenharia - UERJ

Prof. Dr. Orlando Bernardo Filho (Orientador)
Faculdade de Engenharia - UERJ

Prof^a. Dra. Maria Luiza Fernandes Velloso
Faculdade de Engenharia - UERJ

Prof. Dr. João Baptista de Oliveira e Souza Filho
Departamento de Engenharia Elétrica – CEFET-RJ

Rio de Janeiro

2013

DEDICATÓRIA

Dedico este trabalho aos meus Pais, que acreditam com fé nos meus sonhos, transmitindo paciência, carinho e amor, e que respeitaram em silêncio meus muitos momentos de ausência e destempero.

AGRADECIMENTOS

A Deus, que está acima de todas as coisas deste vasto mundo, concebendo sempre nossos desejos e vontades, mesmo quando de forma oculta ou improvável.

Ao meu orientador, professor Nival Nunes, que me aceitou de imediato, possibilitando esta oportunidade de trabalharmos juntos, pela dedicação em suas orientações durante o transcorrer da pesquisa; sempre com uma palavra amiga e doce de incentivo, mesmo nos meus momentos mais difíceis.

Ao meu orientador, professor Orlando Bernardo, por ter embarcado comigo neste projeto, pelas sábias orientações no desenvolvimento e elaboração desta dissertação, e pela enorme paciência com minhas dúvidas e anseios. Nunca houve um dia em que não vencêssemos juntos.

Ao Richard, que me ajudou a remar em direção ao norte, com muito bom humor e amizade.

Aos professores e colegas do PEL que passaram pela minha vida e, de algum modo, colaboraram, direta ou indiretamente, com esta pesquisa.

A UERJ, a Faculdade de Engenharia, ao Departamento de Pós-Graduação em Engenharia Eletrônica e ao Arthur Kowarski.

Aos meus companheiros de trabalho da Rede Globo, que sempre me ajudaram muito com ideias, sugestões e trocas de horário no plantão.

A minha mãe, que acordava de madrugada para deixar minha mesa de estudos recheada de guloseimas e chá verde durante a elaboração da dissertação.

Ao meu pai, que nunca mediu esforços, desmarcando qualquer compromisso para garantir que eu sempre chegasse à UERJ na hora.

A minha querida irmã, que me incentivou com muito amor e broncas, mesmo de longe.

A minha avó, matriarca da família, que abdicou do mundo acadêmico quando emigrou da China para o Brasil com o bolso vazio de dinheiro, mas repleto de sonhos a construir.

Em especial, à Scintilla, amiga de todas as horas e uma das maiores incentivadoras da finalização deste projeto.

RESUMO

JI, Carolina Yoshico. *Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação em nuvem*. 2013. 231f. Dissertação (Mestrado em Engenharia Eletrônica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

O objetivo deste trabalho é avaliar os riscos de ocorrências de intrusos em um sistema de computação em nuvem para sistemas distribuídos utilizando lógica nebulosa. A computação em nuvem é um tema que vem sendo bastante abordado e vem alavancando discussões calorosas, tanto na comunidade acadêmica quanto em palestras profissionais. Embora essa tecnologia esteja ganhando mercado, alguns estudiosos encontram-se céticos afirmando que ainda é cedo para se tirar conclusões. Isto se deve principalmente por causa de um fator crítico, que é a segurança dos dados armazenados na nuvem. Para esta dissertação, foi elaborado um sistema distribuído escrito em Java com a finalidade de controlar um processo de desenvolvimento colaborativo de *software* na nuvem, o qual serviu de estudo de caso para avaliar a abordagem de detecção de intrusos proposta. Este ambiente foi construído com cinco máquinas (sendo quatro máquinas virtuais e uma máquina real). Foram criados dois sistemas de inferência nebulosos, para análise de problemas na rede de segurança implementados em Java, no ambiente distribuído. Foram realizados diversos testes com o intuito de verificar o funcionamento da aplicação, apresentando um resultado satisfatório dentro dessa metodologia.

Palavras-chave: Detecção de intrusos; Lógica nebulosa; Sistemas distribuídos; Computação em nuvem; Desenvolvimento de *software*.

ABSTRACT

Jl, Carolina Yoshico. *A fuzzy system intrusion detection for cloud computing*. 2013. 231f. Dissertação (Mestrado em Engenharia Eletrônica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

The objective of this study is to evaluate the risk of occurrence of intruders in a system of cloud computing at distributed systems using fuzzy logic. Cloud computing is a topic that has been widely discussed and has been leveraging heated discussions, both in academic and in professional speaking. Although this technology is gaining market share, some academics are incredulous saying that is too early to draw conclusions. This is mainly because of a critical factor, which is the security of data stored in the cloud. For this thesis, we designed a distributed system written in Java, with the purpose of controlling a process of software's development in the cloud, which served as a case study to evaluate the approach proposed intrusion detection. This environment was built with five machines (being four virtual machines and one real machine). It was created two fuzzy inference systems for analysis of problems in network security implemented in Java, in the distributed environment. Several tests were performed in order to verify the functionality of the application, presenting a satisfactory outcome within this methodology.

Keywords: Analysis of intrusion detection systems; Fuzzy logic; Cloud computing; Software development.

LISTA DE FIGURAS

FIGURA 1 - REPRESENTAÇÃO MODULAR DE UM SIN (BERNARDO FILHO, 1999).....	25
FIGURA 2 – VALOR CALCULADO PELO MÉTODO DO CENTROIDE, ADAPTADO DE (BERNARDO FILHO, 1999).	27
FIGURA 3 - MODELO EM CASCATA, ADAPTADO DE (PRESSMAN, 2006).	29
FIGURA 4- MODELO INCREMENTAL, ADAPTADO DE (PRESSMAN, 2006).....	30
FIGURA 5- PARADIGMA DA PROTOTIPAGEM, ADAPTADO DE (PRESSMAN, 2006).	31
FIGURA 6 - FASES DO PU, ADAPTADO DE (PRESSMAN, 2006).....	34
FIGURA 7 - MODELO OSI.	37
FIGURA 8 - O MODELO DE REFERÊNCIA OSI (TANEMBAUM, 2011).	37
FIGURA 9 – ARQUITETURA DE PROTOCOLOS DA INTERNET (BERNARDO FILHO, 1999).....	40
FIGURA 10 - MODELOS DE SERVIÇOS (SOUSA, 2010).....	56
FIGURA 11 – ARQUITETURA DO SISTEMA DISTRIBUÍDO PARA DESENVOLVIMENTO DE <i>SOFTWARE</i> NA NUVEM.....	64
FIGURA 12 – ARQUITETURA DO GESTOR DE <i>SOFTWARE</i> NA NUVEM.....	66
FIGURA 13 - MODELO DE CASO DE USO PARA O GESTOR DE <i>SOFTWARE</i>	68
FIGURA 14- MODELO DE CLASSES DO GESTOR DE <i>SOFTWARE</i> NA NUVEM	70
FIGURA 15 – ARQUITETURA DO NODO DE <i>SOFTWARE</i> NA NUVEM.	71
FIGURA 16 - MODELO DE CASOS DE USO DO NODO DE <i>SOFTWARE</i> NA NUVEM	73
FIGURA 17 - MODELO DE CLASSES DO NODO DE <i>SOFTWARE</i> NA NUVEM.....	74
FIGURA 18 – MÁQUINA DE ESTADOS DO PROTOCOLO DO GESTOR DE <i>SOFTWARE</i> NA NUVEM.	75
FIGURA 19 – MÁQUINA DE ESTADOS DO PROTOCOLO DO NODO DE <i>SOFTWARE</i> NA NUVEM.....	76
FIGURA 20 – SISTEMAS DE INFERÊNCIA NEBULOSOS PARA DETECÇÃO DE INTRUSOS NA NUVEM	78
FIGURA 21 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA ATRASO.	79
FIGURA 22 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA TENTATIVAS.	79
FIGURA 23 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA SERVIÇOS.....	80
FIGURA 24– FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA IPR.....	80
FIGURA 25 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA AFP.....	81
FIGURA 26 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA ACE.....	82
FIGURA 27 – FUNÇÕES DE PERTINÊNCIA DOS TERMOS DA VARIÁVEL LINGUÍSTICA IPS.	82
FIGURA 28 – ALGORITMO DE MONITORAMENTO DO GESTOR DE <i>SOFTWARE</i> NA NUVEM	103
FIGURA 29 – INTERFACE DO GESTOR DE <i>SOFTWARE</i> NA NUVEM.	104
FIGURA 30 – OPÇÕES DO MENU GERENCIAR.	104
FIGURA 31 – ENTRADA DE DADO: DEFINA A MINHA PORTA.....	105
FIGURA 32 – CALIBRAR.....	105
FIGURA 33 – MANTER BANCO DE DADOS.	106
FIGURA 34 – TABELA A SER MANTIDA.	106
FIGURA 35 – TABELA A SER MANTIDA: SISTEMA.	107
FIGURA 36 – TABELA A SER MANTIDA: EQUIPE.....	108
FIGURA 37 – TABELA A SER MANTIDA: FASE.	108
FIGURA 38 – TABELA A SER MANTIDA: ATIVIDADE.	109
FIGURA 39 – TABELA A SER MANTIDA: HOST.....	110

FIGURA 40 – TABELA A SER MANTIDA: SERVIÇO.	111
FIGURA 41 – TABELA A SER MANTIDA: MONITORAMENTO.	111
FIGURA 42 – OPÇÕES DO CAMPO MONITORAR.	112
FIGURA 43 – MONITORAR: INICIAR MONITORAMENTO.	112
FIGURA 44 – EXEMPLO DE MONITORAÇÃO.	113
FIGURA 45 – CANCELAR MONITORAMENTO.	113
FIGURA 46 – CAIXA DE DIÁLOGO DO MONITORAMENTO.	114
FIGURA 47 – MENU AJUDA: SOBRE.	114
FIGURA 48 – SOBRE: GESTOR DE <i>SOFTWARE</i> NA NUVEM – VERSÃO 1.0.	114
FIGURA 49 – MENUS DO <i>SOFTWARE</i> NODO DO <i>SOFTWARE</i> NA NUVEM.	117
FIGURA 50 – MENUS DO <i>SOFTWARE</i> NODO DO <i>SOFTWARE</i> NA NUVEM.	117
FIGURA 51 – ENTRADA DE DADO: DEFINA A MINHA PORTA.	118
FIGURA 52 – ENTRADA DE DADO: DEFINA A MINHA PORTA.	118
FIGURA 53 – OPÇÕES DO MENU IDENTIFICAÇÃO DO GESTOR.	119
FIGURA 54 – ENTRADA DE DADO: DEFINA A MINHA PORTA.	119
FIGURA 55 – ENTRADA DE DADO: ENDEREÇO IP DO GESTOR.	119
FIGURA 56 – MENU AJUDA: SOBRE.	120
FIGURA 57 – SOBRE: VERSÃO 1.0.	120
FIGURA 58 - PROBLEMA DE SEGURANÇA DETECTADO.	124
FIGURA 59 - CONTEÚDO DA TABELA HOST ANTES DA CALIBRAÇÃO.	125
FIGURA 60 - CONTEÚDO DA TABELA HOST APÓS A CALIBRAÇÃO.	125
FIGURA 61- INÍCIO DO NODO ENVIANDO ARTEFATO.	126
FIGURA 62 - TÉRMINO DO ENVIO DO ARTEFATO.	126
FIGURA 63 - ARTEFATO ENVIADO.	127
FIGURA 64 - ARTEFATO ENVIADO PELO NODO RECEBIDO COM ÊXITO PELO GESTOR.	127
FIGURA 65- DIRETÓRIO DO GESTOR CONTENDO OS ARTEFATOS ENVIADOS PELO NODO.	128
FIGURA 66 - MANTER ATIVIDADE.	128
FIGURA 67 - MANTER EQUIPE.	129
FIGURA 68 - DEFININDO PORTA DO GESTOR.	129
FIGURA 69 - DEFININDO IP DO GESTOR.	130
FIGURA 70 – ITERAÇÃO 2 DO HOST.	130
FIGURA 71 - TABELA HOST ANTES DA CALIBRAÇÃO.	131
FIGURA 72: TABELA SERVIÇOS.	132
FIGURA 73: TABELA HOST APÓS CALIBRAÇÃO.	132

LISTA DE TABELAS

TABELA 1- PROPRIEDADES ALGÉBRICAS.	23
TABELA 2 – REGRAS DE INFERÊNCIA NEBULOSAS DO SIR.....	83
TABELA 3 – REGRAS DE INFERÊNCIA NEBULOSAS DO SIS.	84
TABELA 4 - MÉTODOS DA CLASSE INTERFACEMANTERATIVIDADE.JAVA.	89
TABELA 5 - MÉTODOS DA CLASSE INTERFACEMANTEREQUIPE.JAVA.	89
TABELA 6 - MÉTODOS DA CLASSE INTERFACEMANTERFASE.JAVA.....	90
TABELA 7 - MÉTODOS DA CLASSE INTERFACEMANTERHOST.JAVA.	90
TABELA 8 - MÉTODOS DA CLASSE INTERFACEMANTERMONITORAMENTO.JAVA.....	91
TABELA 9 - MÉTODOS DA CLASSE INTERFACEMANTERSERVICO.JAVA.	91
TABELA 10 - MÉTODOS DA CLASSE INTERFACEMANTERSISTEMA.JAVA.	92
TABELA 11 - MÉTODOS DA CLASSE CLIENTE.JAVA.	93
TABELA 12 - MÉTODOS DA CLASSE PROTOCOLO.JAVA.	93
TABELA 13 - MÉTODOS DA CLASSE SERVIDOR.JAVA.....	93
TABELA 14 - MÉTODOS DA CLASSE SERVIDORTHREAD.JAVA.	93
TABELA 15 - MÉTODOS DA CLASSE GESTORSOFT.JAVA.....	94
TABELA 16 - MÉTODOS DA CLASSE MONITORNODOSTHREAD.JAVA.	95
TABELA 17 - MÉTODOS DA CLASSE ATIVIDADEDAO.JAVA.	96
TABELA 18 - MÉTODOS DA CLASSE EQUIPEDAO.JAVA.	97
TABELA 19 - MÉTODOS DA CLASSE FASEDAO.JAVA.	97
TABELA 20 - MÉTODOS DA CLASSE HOSTDAO.JAVA.	97
TABELA 21 - MÉTODOS DA CLASSE MONITORAMENTODAO.JAVA.	97
TABELA 22 - MÉTODOS DA CLASSE SERVICODAO.JAVA.	98
TABELA 23 - MÉTODOS DA CLASSE SISTEMADAO.JAVA.	98
TABELA 24 - MÉTODOS DA CLASSE CLIENTE.JAVA.	115
TABELA 25 - MÉTODOS DA CLASSE PROTOCOLO.JAVA.	115
TABELA 26 - MÉTODOS DA CLASSE SERVIDOR.JAVA.....	116
TABELA 27 - MÉTODOS DA CLASSE CONTROLANODO.JAVA.	116
TABELA 28 - RESULTADOS DO SIR EXECUTADOS PELO MATLAB.	121
TABELA 29 - RESULTADOS OBTIDOS PELO GESTOR.....	122
TABELA 30 - RESULTADOS DO SIS EXECUTADOS PELO MATLAB.....	123
TABELA 31- RESULTADOS OBTIDOS PELO GESTOR.	123
TABELA 32 - ALOCAÇÃO DE IPS PARA OS NODOS E GESTOR.....	124
TABELA 33 – TABELA MONITORAMENTO NO HEIDISQL.	131
TABELA 34 - MONITORAMENTO DO HEIDISQL.....	132
TABELA 35 - ATAQUES SIMULADOS.	133
TABELA 36 - RESULTADOS.	133

LISTA DE ABREVIATURAS E SIGLAS

IDS	<i>Intrusion Detection System</i>
SDI	Sistema de Detecção de Intrusos
SIN	Sistema de Inferência Nebuloso
UML	<i>Unified Modeling Language</i>
API	<i>Application Programming Interface</i>
IA	Inteligência Artificial
CRM	<i>Customer Relationship Management</i>
OSI	<i>Open Systems Interconnection</i>
ISO	<i>International Organization for Standardization</i>
TI	Tecnologia da Informação
SIR	Sistema de Inferência de Rede
SIS	Sistema de Inferência de Segurança
IPR	Índice de Problema de Rede
IPS	Índice de Problema de Segurança
AFP	Artefatos Fora do Prazo
ACE	Artefatos com Erro
MVC	<i>Model, View, Controller</i>
SSO	<i>Single Sign-On</i>
PDF	<i>Portable Document Format</i>
CBSF	Segundo Congresso Brasileiro de Sistemas Fuzzy
OO	Orientado a Objetos
PU	Processo Unificado

LISTA DE SÍMBOLOS

MB	Megabyte
ms	milissegundos

SUMARIO

INTRODUÇÃO	16
MOTIVAÇÃO.....	17
OBJETIVOS	17
CONTRIBUIÇÃO.....	18
ORGANIZAÇÃO DO TRABALHO	19
1 FUNDAMENTOS TEÓRICOS.....	20
1.1 Lógica.....	20
1.1.1 <u>Conjuntos Nebulosos</u>	20
1.1.2 <u>Operações com Conjuntos Nebulosos</u>	21
1.1.3 <u>Variáveis Linguísticas</u>	23
1.1.4 <u>Funções de Implicação ou Proposição Nebulosa</u>	24
1.1.5 <u>Processo de Agregação</u>	25
1.1.6 <u>Sistemas de Inferência Nebulosos (SIN)</u>	25
1.2 Processos de Desenvolvimento de <i>Software</i>	27
1.2.1 <u>Modelos de Processos de <i>Software</i></u>	29
1.2.2 <u>Unified Modeling Language (UML)</u>	33
1.3 Redes de Computadores.....	35
1.3.1 <u>O Modelo de Referência OSI</u>	36
1.3.2 <u>Arquitetura de Protocolos da Internet</u>	39
1.4 Segurança	41
1.4.1 <u>Vulnerabilidades</u>	43
1.4.2 <u>Ameaças</u>	44
1.4.3 <u>Mecanismos de Segurança</u>	45
1.4.4 <u>Técnicas de Invasão e Descoberta de Informações</u>	48
1.4.5 <u>Sistema de Detecção de Intrusos</u>	49
1.5 Computação em Nuvem	51
1.5.1 <u>Modelos de Implantação na Nuvem</u>	52
1.5.2 <u>Segurança na Nuvem</u>	53
1.5.3 <u>Modelos de Serviços</u>	55
1.5.4 <u>Comentários</u>	58
2 TRABALHOS RELACIONADOS.....	59
2.1 <u>Introdução</u>	59
2.2 <u>Computação em Nuvem</u>	59
2.3 <u>Sistemas de Detecção de Intrusos</u>	60
2.4 <u>Segurança de Rede</u>	61
2.5 <u>Mecanismos de Segurança com Lógica Nebulosa</u>	61
2.6 <u>Gestão de Projetos com Lógica Nebulosa</u>	62
2.7 <u>Comentários</u>	62
3 MODELAGEM.....	63
3.1 <u>Introdução</u>	63

3.2	Arquitetura do Estudo de Caso	63
3.3	Análise do Gestor de <i>Software</i> na Nuvem	65
3.3.1	Modelo de Casos de Uso	67
3.3.2	Modelo de Classes	69
3.4	Análise do Nodo de <i>Software</i> na Nuvem	71
3.4.1	Modelo de Casos de Uso	72
3.4.2	Modelo de Classes	73
3.5	Protocolo do Gestor de <i>Software</i> na Nuvem	74
3.6	Protocolo do Nodo de <i>Software</i> na Nuvem	75
3.7	Modelagem dos Sistemas de Interferência Nebulosos	76
3.8	Comentários	85
4	IMPLEMENTAÇÃO	86
4.1	Introdução	86
4.2	Projeto Físico do Banco de Dados	86
4.3	Especificação dos Métodos do Gestor de <i>Software</i> na Nuvem	88
4.4	Algoritmo de Inferência Nebuloso	99
4.5	Algoritmo de Monitoramento do Gestor de <i>Software</i> na Nuvem	100
4.6	Interfaces do Gestor de <i>Software</i> na Nuvem	103
4.7	Especificação dos Métodos do Nodo de <i>Software</i> na Nuvem	115
4.8	Interfaces do Nodo de <i>Software</i> na Nuvem	116
4.9	Comentários	120
5	TESTES E RESULTADOS	121
5.1	Introdução	121
5.2	Testes com o Sistema de Inferência Nebuloso de Rede	121
5.3	Testes com o Sistema de Inferência Nebuloso de Segurança	122
5.4	Ambiente de Teste da Computação em Nuvem	124
5.5	Testes com o Gestor e um Nodo de <i>Software</i> na Nuvem	125
5.6	Testes com o Gestor e Vários Nodos de <i>Software</i> na Nuvem	131
5.7	Comentários	134
6	CONCLUSÕES	135
	REFERÊNCIAS	138
	APÊNDICE A — FONTES DO GESTOR DE SOFTWARE NA NUVEM	143
A.1	PACOTE CONTROLE	143
A.2	PACOTE DAO	156
A.3	PACOTE INTERFACE_REDE	175
A.4	PACOTE INTERFACE_USUARIO	181
A.5	PACOTE MODELO	207
	APÊNDICE B — FONTES DO NODO DE SOFTWARE NA NUVEM	222
B.1	PACOTE CONTROLE	222
B.2	PACOTE INTERFACE_REDE	223
B.3	PACOTE INTERFACE_USUARIO	228
B.4	PROTOCOLO MODELO	230

INTRODUÇÃO

O aumento da necessidade da utilização das redes de computadores acarreta diretamente no aumento da complexidade dos sistemas integrados de gerência. Isto ocorre porque cada vez mais novos dispositivos são inseridos na rede com a necessidade de comunicação entre eles e, com isso, a tarefa de gerenciá-la (BLACK, 2008) está se tornando cada vez mais importante e crucial para o seu desempenho.

A computação em nuvem (RUSCHEL, 2010) vem auxiliando neste processo da evolução tecnológica, mas com ela também vieram os problemas de segurança do compartilhamento de informações. A computação em nuvem é uma forma de processamento da informação onde os recursos de informática são enormemente escaláveis e são fornecidos aos clientes como serviços através da Internet. Trata-se de uma solução *all-inclusive* na qual componentes tais como *hardware*, *software*, redes, armazenamento, etc. são fornecidos aos usuários rapidamente e por demanda. As funcionalidades compartilhadas são gerenciadas para garantir alta disponibilidade, segurança e qualidade.

Os recursos são oferecidos como serviços na Internet, onde os usuários transferem seus dados e aplicações para a “nuvem”, podendo acessá-los de qualquer local. Esta tecnologia caracteriza-se por ser redundante e resiliente. Com o crescimento e complexidade da rede, executar uma verificação de riscos para prover segurança aos dados e informações presentes é uma tarefa imprescindível. Analisar o tráfego da rede, bem como estudar e averiguar seu comportamento (CARDANA, 2006), são maneiras de garantir a sua integridade, prevenindo-a de ataques e eventos suspeitos. Para descobrir se uma rede está desprotegida ou vulnerável a acessos não autorizados, é necessária a implementação de um sistema de detecção de intrusos, também conhecido como IDS (do inglês, *Intrusion Detection System*) (PTACEK, 1998) ou SDI (Sistema de Detecção de Intrusos).

Recentemente, no dia 26 de Novembro de 2012, alguns serviços do *Google* ficaram fora do ar por aproximadamente vinte minutos (embora outros relatos informem queda maior do que trinta minutos), o suficiente para repercutir mundialmente (INFO, 2012). Organizações que utilizam o *Google Apps*, ferramenta da empresa que gerencia todos os *e-mails* de uma companhia, ficaram sem o serviço de correio eletrônico durante a falha. Este episódio salientou a alta dependência não apenas do brasileiro, mas também de praticamente todos os demais países em relação aos serviços oferecidos pelo *Google*. Em nota oficial, a assessoria

da organização afirmou que todos os serviços estavam funcionando como o esperado e sem problemas. Assim, ficou clara também a importância da evidência da fragilidade do serviço. Uma questão cabe ser colocada: até que ponto é possível confiar plenamente na nuvem?

MOTIVAÇÃO

Grandes projetos de desenvolvimento de *software* demandam a criação de grandes quantidades de artefatos e ainda a interação e integração de muitas atividades e muitas pessoas, tornando a coordenação dos processos e equipes um processo extremamente importante e complexo. O trabalho cooperativo, de forma distribuída, tem sido utilizado de forma crescente, principalmente no desenvolvimento de grandes e complexos projetos de *software*.

De fato, neste cenário de desenvolvimento de *software* é de suma importância a utilização de ferramentas, técnicas e metodologias que auxiliem equipes fisicamente distribuídas na execução de atividades e, também, que possibilitem aos gestores o monitoramento, controle, e a intervenção em todo processo, se necessário for.

A motivação para este trabalho foi estudar a possibilidade de minimizar a vulnerabilidade dos riscos de rede e segurança na nuvem, com a aplicação da proposta de desenvolvimento de *software* na nuvem. A integridade dos dados é de fundamental importância e os sistemas de informação são passíveis de falha física e humana e ainda alvo em potencial de ataques maliciosos para a coleta de dados. Com o presente estudo, espera-se contribuir de forma significativa para a detecção de intrusos por meio da utilização da lógica nebulosa, diminuindo ao máximo a necessidade da interferência humana no processo.

OBJETIVOS

A proposta desta dissertação foi implementar um SDI modelado com lógica nebulosa para a análise de riscos em uma nuvem, capaz de detectar invasões ou comportamentos maliciosos de acordo com comparações realizadas com as regras na base de dados. Para a validação desse modelo, foi necessário o desenvolvimento de um ambiente com a criação das regras, onde foi implementado em Java. Para criação da rede de computadores utilizada nesta simulação, foi utilizado o VirtualBox (VIRTUALBOX, 2012). Os códigos do sistema foram escritos utilizando a IDE Eclipse e o banco de dados no HeidiSQL (HEIDISQL, 2012).

CONTRIBUIÇÃO

Durante a elaboração desta pesquisa foi escrito um artigo apresentado no II CBSF - Segundo Congresso Brasileiro de Sistemas *Fuzzy* (JI, 2012) em Novembro de 2012 no Rio Grande do Norte, o qual serviu de embasamento para a construção do estudo de caso exposto nesta dissertação.

ORGANIZAÇÃO DO TRABALHO

No **Capítulo 1** são introduzidos os fundamentos teóricos que auxiliam no entendimento desta dissertação, tais como os conceitos de lógica nebulosa, processos de desenvolvimento de *software*, redes de computadores e computação em nuvem.

Em seguida, no **Capítulo 2**, são discutidos alguns trabalhos relacionados ao tema proposto.

No **Capítulo 3**, descreve-se o processo de modelagem adotado neste projeto. Aqui são descritos parâmetros da arquitetura proposta, bem como o protocolo utilizado. As regras dos sistemas de inferências nebulosas são ilustradas neste capítulo.

No **Capítulo 4**, apresenta-se a implementação adotada neste trabalho, com o projeto físico do banco de dados, e os algoritmos de inferência nebulosa e do gestor de *software*. As interfaces do gestor de *software* são esclarecidas neste capítulo.

No **Capítulo 5**, são impressos os resultados dos testes e simulações do projeto, ou seja, testes para ambos os sistemas de inferência nebuloso, tanto da rede quanto o de segurança. São apresentados ainda os demais testes com o protocolo escolhido no terceiro capítulo e com o gestor de *software* apresentado no capítulo anterior.

No **Capítulo 6**, por fim, apresentam-se as conclusões desta dissertação e a relação de trabalhos futuros que o tema do presente estudo proporciona.

1 FUNDAMENTOS TEÓRICOS

Neste capítulo, serão apresentados os fundamentos teóricos dos tópicos abordados nesta dissertação, proporcionando uma visão geral de todos os conceitos necessários para o entendimento do escopo desta dissertação.

1.1 Lógica

Original da etimologia grega, a palavra lógica significa pensamento, ideia ou razão. É uma ciência baseada profundamente na matemática e com raízes intrinsecamente ligadas à filosofia. A lógica é responsável por organizar as regras de modo que garanta a verdade em seu resultado final. O filósofo grego Aristóteles (384-322 a.C.) criou a ciência da lógica, e o seu primeiro estudo formal, estabelecendo um conjunto de regras para que conclusões pudessem ser validadas dentro de um determinado sistema lógico. A teoria diz que todo raciocínio lógico é fundamentado em premissas e conclusões, de modo a conferir valores “verdade” às afirmativas; classificando-as como verdadeiras ou falsas (NEO, 2008).

1.1.1 Conjuntos Nebulosos

A ideia de conjuntos nebulosos foi introduzida por Lotfi Zadeh (ZADEH, 1965) como uma alternativa para tratar, de maneira formal, problemas de caráter subjetivo que tipicamente apresentem informações vagas e imprecisas, que necessitem de um raciocínio aproximado.

Seja X um universo de discurso, no qual representa-se um elemento genérico por x , e um determinado conjunto é designado por A . Na teoria clássica de conjuntos, um elemento possui caráter binário: o elemento pertence ($x \in A$) ou não pertence ($x \notin A$) a este conjunto.

Através de uma função característica f_A é possível definir esta propriedade, fazendo o mapeamento dos elementos do universo de discurso no conjunto binário $\{0, 1\}$, conforme apresentado na Equação 1:

$$f_A(x): X \rightarrow \{0,1\} \quad f_A(x) = \begin{cases} 1 & \text{se e somente se } x \in A \\ 0 & \text{se e somente se } x \notin A \end{cases} \quad (1)$$

1.1.2 Operações com Conjuntos Nebulosos

Nos conjuntos nebulosos, assim como nos conjuntos *crisp* (clássicos), são necessárias algumas operações de interseção, união e negação, entre outras, que serão mostradas a seguir.

Operadores

Zadeh (ZADEH, 1965) fez uso das funções de máximo e mínimo, respectivamente com o intuito de definir as operações de união e interseção. Por exemplo, o mínimo ou o máximo de dois elementos, a e b é definido pelas Equações 1 e 2.

$$\begin{aligned} a \wedge b &= \min(a, b) = a \text{ se } a \leq b \\ &= b \text{ se } a > b \end{aligned} \quad (1)$$

$$\begin{aligned} a \vee b &= \max(a, b) = a \text{ se } a \geq b \\ &= b \text{ se } a < b \end{aligned} \quad (2)$$

Sejam A e B subconjuntos nebulosos de X . Sua união (ver Equação 3) é um subconjunto nebuloso $A \cup B$, onde " \vee " é utilizado para representar uma disjunção lógica (OLIVEIRA, 1999).

$$(A \cup B)(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x), \forall x \in X \quad (3)$$

Sejam A e B subconjuntos difusos de X . Sua interseção (Equação 4) é um subconjunto nebuloso $A \cap B$, onde " \wedge " representa uma conjunção lógica (OLIVEIRA, 1999).

$$(A \cap B)(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x), \forall x \in X \quad (4)$$

Seja A um subconjunto nebuloso de X . A Negação (que também é denominado Complemento) de A denotado por $neg(A)$ é o conjunto nebuloso definido pela Equação 5 (OLIVEIRA, 1999).

$$Neg(A) = X - A \text{ ou } (neg(A))(x) = 1 - \mu_A(x), \forall x \in X \quad (5)$$

Sejam A e B subconjuntos nebulosos de X, o produto algébrico de A e B, representado por AB é definido pela Equação 6 (FONSECA, 2003) a seguir.

$$\mu_{AB}(x) = \mu_A(x)\mu_B(x), \forall x \in X \quad (6)$$

Sejam A e B subconjuntos nebulosos de X, a soma algébrica de A e B (Equação 7), representada por $A \oplus B$ é definida pela soma das suas funções de pertinência, subtraindo o seu produto algébrico. Com isso, temos que:

$$\mu_{A \oplus B}(x) = \mu_A(x) + \mu_B(x) - \mu_{AB}(x), \forall x \in X \quad (7)$$

Operações de união e interseção são casos característicos de situações mais abrangentes de associação de conjuntos nebulosos.

Propriedades Algébricas

Através do emprego das definições de união, interseção e complemento, é possível verificar que diversas propriedades algébricas dos conjuntos ordinários também se aplicam a conjuntos nebulosos (TANSCHKEIT, 1999).

As propriedades mais utilizadas na operação de sistemas de inferência nebulosos são apresentadas na tabela a seguir:

Tabela 1- Propriedades Algébricas.

NOME DA PROPRIEDADE	EXPRESSÃO
Associatividade	$(\tilde{A}1 \cap \tilde{A}2) \cap \tilde{A}3 = \tilde{A}1 \cap (\tilde{A}2 \cap \tilde{A}3)$ $(\tilde{A}1 \cup \tilde{A}2) \cup \tilde{A}3 = \tilde{A}1 \cup (\tilde{A}2 \cup \tilde{A}3)$
Comutatividade	$\tilde{A}1 \cap \tilde{A}2 = \tilde{A}2 \cap \tilde{A}1$ $\tilde{A}1 \cup \tilde{A}2 = \tilde{A}2 \cup \tilde{A}1$
Distributividade (Lei da Fatoração)	$\tilde{A}1 \cap (\tilde{A}2 \cup \tilde{A}3) = (\tilde{A}1 \cap \tilde{A}2) \cup (\tilde{A}1 \cap \tilde{A}3)$ $\tilde{A}1 \cup (\tilde{A}2 \cap \tilde{A}3) = (\tilde{A}1 \cup \tilde{A}2) \cap (\tilde{A}1 \cup \tilde{A}3)$
Idempotência	$\tilde{A}1 \cap \tilde{A}1 = \tilde{A}1$ $\tilde{A}1 \cup \tilde{A}1 = \tilde{A}1$
Involução (Lei da Dupla Negação)	$\sim(\sim\tilde{A}1) = \tilde{A}1$
Absorção	$\tilde{A}1 \cap (\tilde{A}1 \cup \tilde{A}2) = \tilde{A}1$ $\tilde{A}1 \cup (\tilde{A}1 \cap \tilde{A}2) = \tilde{A}1$
Transitividade	Se $\tilde{A}1 \subset \tilde{A}2$ e $\tilde{A}2 \subset \tilde{A}3$ então $\tilde{A}1 \subset \tilde{A}3$
Lei de DeMorgan	$\sim(\tilde{A}1 \cup \tilde{A}2) = \sim\tilde{A}1 \cap \sim\tilde{A}2$ $\sim(\tilde{A}1 \cap \tilde{A}2) = \sim\tilde{A}1 \cup \sim\tilde{A}2$
Propriedades baseadas nas definições de conjunto vazio e de conjunto universo.	$\tilde{A} \cup \emptyset = \tilde{A}$
	$\tilde{A} \cup X = X$
	$\tilde{A} \cap \emptyset = \emptyset$
	$\tilde{A} \cap X = \tilde{A}$

1.1.3 Variáveis Linguísticas

De acordo com (TANSCHKEIT, 1999), o conceito de uma variável linguística é aquela que apresenta valores que são rótulos de conjuntos nebulosos. Por exemplo, é possível construir um modelo onde os valores de velocidade assumidos sejam baixa, média, alta, etc. Pode-se descrever esses valores através de conjuntos nebulosos. De forma geral, as variáveis podem conter sentenças ou valores em uma linguagem especificada.

Nesse caso, a variável é do tipo linguística. Por exemplo, alguns dos valores da variável nebulosa velocidade poderiam ser expressos por baixa, não baixa, muito baixa, bastante baixa, não muito baixa, baixa mas não muito baixa. Os valores do conjunto nebuloso são sentenças formadas a partir do adjetivo “baixa”, da negação “não”, dos conectivos “e” e “mas” e ainda dos modificadores “muito” e “bastante”.

A principal função das variáveis linguísticas é modelar os fenômenos considerados abstratos, dotados de informações vagas e imprecisas. De modo geral, ao se optar por empregar descrições linguísticas aplicadas por seres humanos, é possível que sistemas muito complexos sejam passíveis de tratamento. O especialista do sistema é o responsável por descrever e quantificar as variáveis linguísticas do sistema.

1.1.4 Funções de Implicação ou Proposição Nebulosa

Uma declaração de implicação nebulosa ou condicional nebulosa descreve uma relação entre variáveis linguísticas. Para ilustrar esta função, consideremos dois conjuntos nebulosos A e B, que contêm valores linguísticos dos universos X e Y, respectivamente. Matematicamente uma declaração nebulosa da forma SE A ENTÃO B é executada como:

$$R : \text{SE A ENTÃO B} = A \rightarrow B = A \times B \quad (8)$$

A relação nebulosa $A \times B$ denota, então, a implicação $A \rightarrow B$ no produto cartesiano dos dois universos $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$ (TANSCHKEIT, 1999).

A base de regras é definida pela combinação de várias regras. São os especialistas do sistema que definem as funções de pertinência e suas regras.

Proposição nebulosa é outro termo que denota função de implicação. Segundo (OLIVEIRA, 1999), a proposição nebulosa propaga relações entre variáveis linguísticas e conjuntos nebulosos, de modo que é possível expor composições por meio de conectivos e transformadores, como no exemplo da velocidade apresentado anteriormente. O processo de avaliação de proposições consiste em estimar o “nível de verdade” ou pertinência apresentado em relação a um problema.

Segundo (TANSCHKEIT, 1999), os operadores de implicação mais utilizados são o mínimo e o produto algébrico. Para sentenças com um antecedente apenas (do tipo SE A ENTÃO B), é possível utilizar as sentenças que seguem abaixo:

$$\text{mínimo: } \mu_B(x, y) = \min(\mu_A(x), \mu_B(y)) = \mu_A(x) \wedge \mu_B(y) \quad (9)$$

$$\text{produto: } \mu_B(x, y) = \mu_A(x) \times \mu_B(y) \quad (10)$$

1.1.5 Processo de Agregação

Agregação é o processo de combinação de conjuntos nebulosos que representam cada regra de saída em um único conjunto de saída. A entrada do processo de agregação é a lista das funções de saída retornadas pelo processo de implicação de cada regra.

O processo de agregação é comutativo, ou seja, a ordem de execução de cada regra não muda o resultado final. Os métodos mais comumente utilizados são o máximo e a soma algébrica, sendo implementados em ferramentas computacionais, como o MatLab, utilizado nesta dissertação também.

1.1.6 Sistemas de Inferência Nebulosos (SIN)

Humanos possuem a aptidão de trabalhar com processos muito complexos, baseados em informações vagas. Os Sistemas de Inferência Nebulosos também são denominados como Controle Nebuloso (CN), e são uma importante aplicação da teoria dos conjuntos nebulosos.

De acordo com (SANDRI, 1999), são localizados na literatura alguns controladores clássicos, como o modelo Mamdani, Larsen (modelo escolhido nesta dissertação), além dos modelos de interpolação Takagi-Sugeno e Tsukamoto. A principal diferença entre os modelos é a forma com a qual são reproduzidos os termos, no que diz respeito à representação das ações de controle e ainda quanto aos operadores empregados para a implementação do controlador em questão.

A Figura 1 apresenta a estrutura básica e genérica de um SIN, propondo a ideia do fluxo de informações envolvido. A seguir tem-se uma breve descrição de cada módulo do sistema.

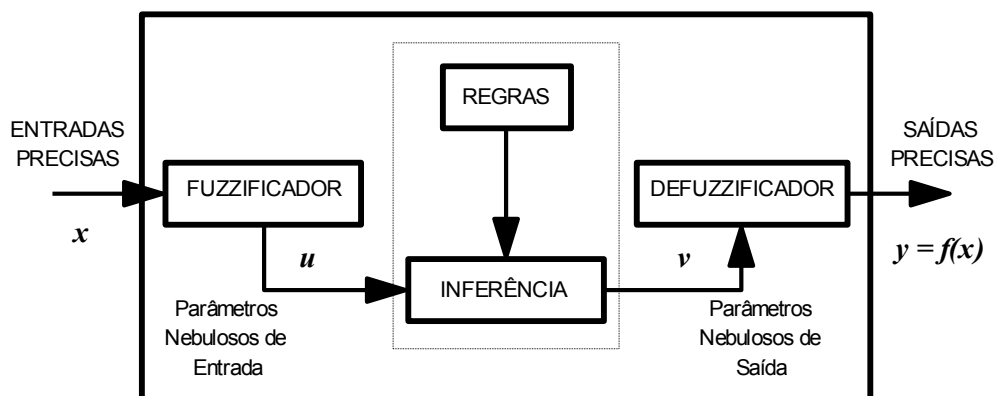


Figura 1 - Representação modular de um SIN (BERNARDO FILHO, 1999).

A Fuzzificação é a primeira etapa da modelagem nebulosa, pois é responsável por transformar as variáveis que se encontram na forma determinística ou "crisp" em um formato nebuloso (pertinência).

Conforme a proposta de lógica nebulosa, o conhecimento é representado através de regras ou proposições. Estas Regras são declarações que relacionam as variáveis do modelo com os conjuntos nebulosos. Isto quer dizer que relacionam os antecedentes com os consequentes.

As regras podem apresentar-se como condicionais ou incondicionais, sendo que sua ordem de execução depende do modelo adotado. A ordem não é importante para os casos de o modelo conter apenas regras condicionais ou incondicionais. No entanto, para o caso de o modelo proposto conter ambos os casos de regra, a ordem de execução é relevante.

As propriedades abaixo são desejáveis para o estabelecimento das regras:

(a) Qualquer combinação das variáveis de entrada deve ativar pelo menos uma regra;

(b) Duas ou mais regras com as mesmas entradas devm ter saídas mutuamente exclusivas. Do contrário, as regras são consideradas inconsistentes.

(c) Não deverão existir regras adjacentes com saídas cujas funções de pertinência não apresentam interseção.

A Inferência Nebulosa avalia as regras que relacionam as variáveis e que levam a conclusão do sistema. O raciocínio é feito através da inferência, a qual possibilita deduzir e concluir partindo de fatos conhecidos (Min-Max). As variáveis linguísticas de entrada e saída representam o conhecimento em inferência nebulosa.

Por fim, tem-se a Defuzzificação, onde as variáveis que estão na forma fuzzificada são transformadas para a forma determinística (crisp), determinando assim o valor real da(s) saída(s). Os métodos mais conhecidos e utilizados recorrentemente para a defuzzificação são o Centro-de-Área (Centróide), o Centro-do-Máximo, a Média-do-Máximo e a Média-Ponderada. O processo de defuzzificação pode ser feito se utilizando três metodologias distintas: valor máximo, média de valores máximos ou o cálculo do centroide. O método do máximo calcula como valor defuzzificado, o ponto do universo de discurso onde a função de pertinência é máxima. Em casos onde a função de pertinência possui mais de um valor máximo, é possível que ocorra certa confusão no cálculo. Com isso, foi proposta a utilização da média dos máximos para se encontrar o valor a ser defuzzificado. Porém, quando a média dos máximos apresenta um valor onde a função de pertinência é zero, isto não é viável. Com

as limitações apresentadas, foi proposto o método do centróide, onde a saída determinística se baseia no cálculo do centro de gravidade do conjunto de suporte de saída, que é mostrado na figura abaixo.

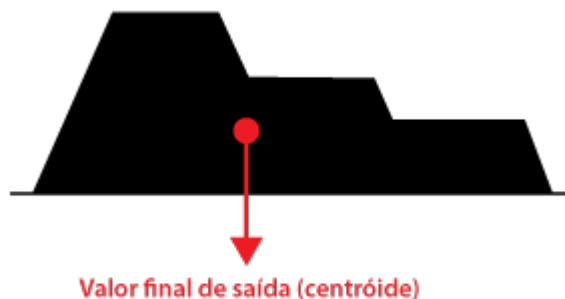


Figura 2 – Valor calculado pelo método do centróide, adaptado de (BERNARDO FILHO, 1999).

O centróide de uma função de pertinência é calculado pelas integrais das funções do termo. As funções de pertinência são segmentos de reta. As integrais representam os somatórios das integrais de cada segmento. Para calcular o centróide do trapézio, é necessário calcular a soma das integrais de cada reta (numerador e denominador) e então, dividir pelo denominador.

A saída escalar é calculada com o somatório dos centroides multiplicado pela área do termo de saída ativados, dividindo-se pela soma das áreas de cada termo. A equação 11 apresenta o respectivo cálculo.

$$\bar{X} = \frac{\sum (\text{Centróide} * \text{Área})}{\sum \text{Área}} \quad (11)$$

O algoritmo utilizando o cálculo do centróide será apresentado com mais detalhes no Capítulo 4.

1.2 Processos de Desenvolvimento de *Software*

À medida que a importância da utilização do *software* se eleva, os desenvolvedores de programas vêm continuamente se empenhando na tentativa do desenvolvimento de tecnologias que facilitem, de forma mais eficaz e menos dispendiosa, a construção e a manutenção de produtos de *software*. Com o surgimento dos primeiros computadores vendidos comercialmente a partir da década de 50, foram criados os primeiros programas a

serem executados nas máquinas. A venda do *software* era feita, muitas vezes, em conjunto com o *hardware*, uma vez que os programas eram totalmente dependentes e vinculados, ou seja, acoplados à arquitetura das máquinas. Atualmente o *software* exerce uma posição de duas vias: ao mesmo tempo em que é produto, é o veículo para a entrega desse. Como produto, ele disponibiliza o potencial do poder computacional presente no *hardware* do computador, ou mais amplamente, na rede de dispositivos conectados. No papel de veículo, por sua vez, o *software* tem a função de alicerce para o controle presente no sistema operacional de cada dispositivo, bem como para a comunicação entre os dispositivos (rede) e para a criação de outros programas (ferramentas e ambientes de *software*). O cenário atual do desenvolvimento de *software* em nada lembra o antigo. Atualmente a indústria de *software* ganhou destaque com a aquisição de equipes especializadas (em prol do programador solitário de antigamente), onde programadores possuem capacidades e habilidades específicas, contribuindo em partes distintas da aplicação.

Muitas definições igualam o termo *software* com programas de computador. No entanto, de acordo com (SOMMERVILLE, 2007), a definição mais ampla onde é a de que os produtos de *software* não são apenas programas, mas também toda parte de toda a documentação associada, juntamente com os dados de configuração, que são necessários para capacitar que esses programas funcionam corretamente. Um sistema de *software* normalmente consiste em uma dada quantidade de programas separados, e os arquivos de configuração, os quais são utilizados para configurar esses programas; documentação de sistema, onde a estrutura do mesmo é descrita, e um manual para o usuário, que explica como usar o sistema e *web-sites* para usuários baixarem as informações sobre o produto.

Basicamente, existem dois tipos fundamentais de *software*:

1. Genéricos: sistemas independentes, que são desenvolvidos por uma organização e vendidos no mercado aberto a qualquer cliente que é capaz de comprá-los.
2. Personalizados (ou sob medida): sistemas que são comissionados por um cliente particular. Um desenvolvedor cria o *software* especialmente para esse cliente, de acordo com as necessidades levantadas.

1.2.1 Modelos de Processos de *Software*

Um processo de *software* é um esqueleto contendo as atividades que são indispensáveis para se construir um *software*. De acordo com (PRESSMAN, 2006), um processo de *software* define a abordagem que é aceita quando o *software* é preparado. A Engenharia de *Software* compreende tecnologias que compõem um processo.

A existência de um processo de *software* não garante sua entrega no prazo acordado, nem que ele satisfaça as necessidades do cliente. É necessário que seja incorporado um padrão ao processo.

Os modelos de processos de *software* comportam modelos que descrevem os conjuntos de elementos de processo. A seguir, serão descritos de forma breve os principais modelos de *software*.

- **Modelo em cascata**

O modelo em cascata (ou também denominado ciclo de vida clássico) é apresentado na Figura 3.

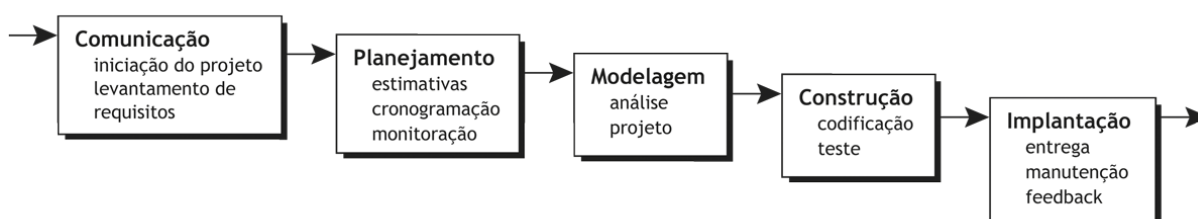


Figura 3 - Modelo em cascata, adaptado de (PRESSMAN, 2006).

Este modelo indica uma investida sistemática e sequencial para o desenvolvimento de produtos de *software*, iniciando com a especificação dos requisitos demandados pelo cliente, evoluindo ao longo do planejamento, modelagem, construção e implantação. Após todas essas etapas, é necessária a manutenção do *software*.

O modelo em cascata é o paradigma mais antigo no campo da Engenharia de *Software*. Porém, nos últimos vinte anos, este modelo tem sido revisto, devido a alguns problemas em sua aplicação, como, por exemplo, a dificuldade de o cliente em explicitar os requisitos necessários. O modelo em cascata precisa das informações mais precisas possíveis e apresenta dificuldade em acomodar a incerteza presente no início dos projetos.

Atualmente o ritmo de projetos de *software* é bem dinâmico e está sujeito a diversas modificações. Com isto, torna-se inviável sua utilização de forma cada vez mais frequente. Mas, ao mesmo tempo, tende a servir como um modelo bastante útil em cenários onde os requisitos e demandas são fixos e o trabalho segue um fluxo linear.

Uma alternativa de modelo é o modelo incremental, como mostra a Figura 4, combinando o modelo em cascata aplicado de maneira iterativa, isto é, aplica sequências lineares de maneira racional, de modo que cada sequência desta produz incrementos do *software*, que são possíveis de serem executados e entregues. Em linhas gerais, o primeiro incremento é denominado de núcleo do produto, que é a parte onde os requisitos básicos são atendidos. Este núcleo é utilizado pelo cliente. Com o resultado deste núcleo, é desenvolvido um plano, que objetiva atualizações do núcleo para atender as necessidades do cliente, preparando todas as características adicionais. Este processo é repetido quantas vezes forem necessárias, de modo que termine somente quando o produto estiver finalizado.

Este modelo tem como principal finalidade apresentar um produto operacional para cada incremento. Este tipo de desenvolvimento incremental é muito útil quando não há mão de obra disponível para a execução completa do projeto dentro do prazo acordado. É fato que os primeiros incrementos podem ser realizados com o tamanho de equipe menor, podendo haver necessidade de adição de mão de obra para os próximos incrementos.

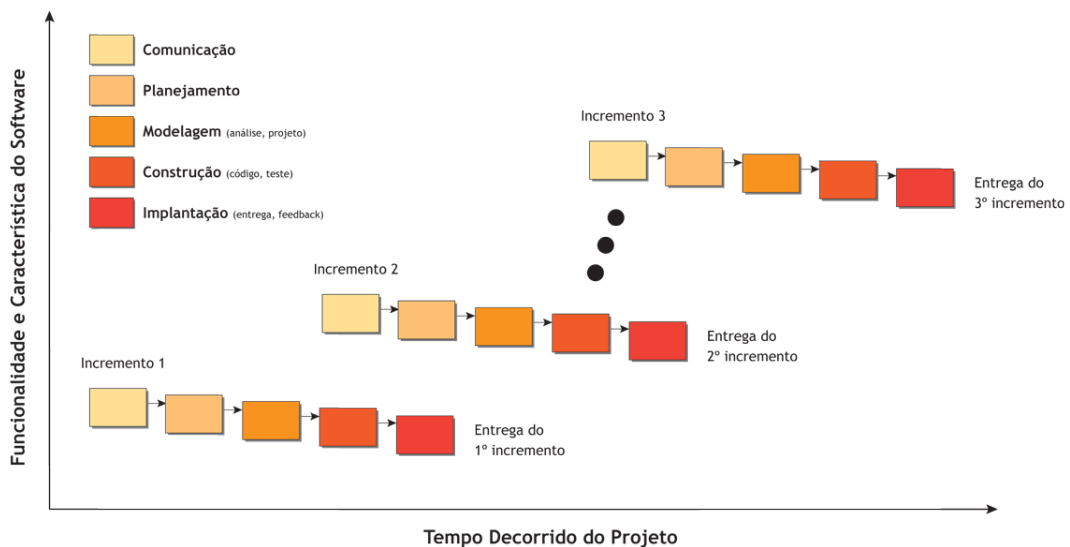


Figura 4- Modelo incremental, adaptado de (PRESSMAN, 2006).

- **Modelos evolucionários**

Um outro tipo de modelo é o modelo evolucionário de processo de *software*. Como o *software* possui a capacidade de evoluir na linha do tempo, suas demandas e necessidades também acompanham estas mudanças, dificultando a finalização do produto. Por isto, a equipe responsável por processos de *software* precisa de um modelo de processo que tenha sido projetado para atender um produto que muda suas características com o passar do tempo.

Os modelos evolucionários são iterativos, capacitando os engenheiros de *software* a construir programas cada vez mais completos e próximos da versão final.

Em caso de o cliente não explicitar as demandas de entrada, processamento e saída para o desenvolvimento, pode-se utilizar o recurso do paradigma da prototipagem, ajudando o engenheiro de *software* e o cliente a compreenderem as necessidades do projeto em conjunto. A prototipagem atua como um mecanismo para a identificação dos requisitos de *software*.

A figura 5 apresenta o modelo da prototipagem, o qual possui como seu primeiro ponto, a comunicação. Esta característica define o início do processo, onde o desenvolvedor e o cliente se encontram para expor, entender e delimitar as necessidades do projeto. É desenhado um plano rápido com a modelagem de acordo com a iteração de prototipagem. Com base nas informações obtidas, é feita a construção do protótipo, que após desenvolvido, é implantado, entregue e avaliado pelo cliente. Com base na avaliação e testes feitos pelo cliente, é reportado o *feedback*, de modo a ajustar o que for necessário.

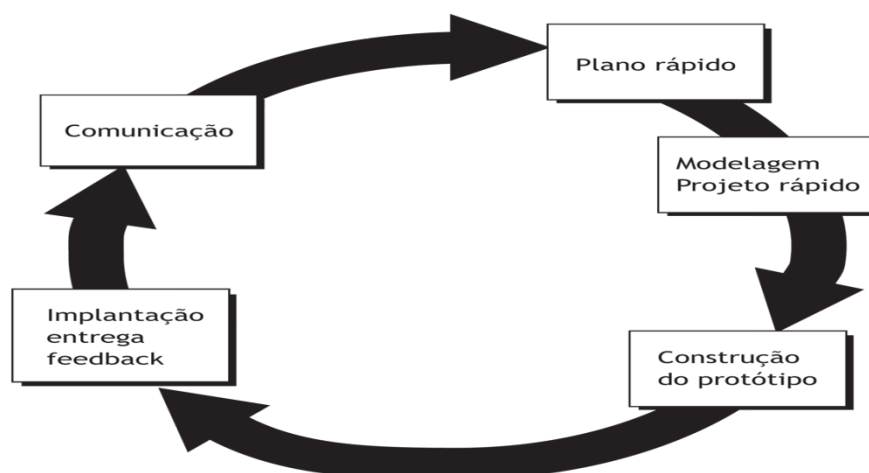


Figura 5- Paradigma da prototipagem, adaptado de (PRESSMAN, 2006).

Apesar de não ser infalível, o modelo de prototipagem vem se apresentando-se como um paradigma efetivo para a Engenharia de *Software*. É fundamental que demanda e desenvolvimento estejam de acordo, desde o início da construção do protótipo. Com isto, o protótipo será descartando, dando lugar ao *software* final.

- **Processo Unificado (PU)**

Segundo (JACOBSON, 1999):

Hoje, a tendência em software é em direção a sistemas maiores, mais complexos. Isso se deve, em parte, ao fato de que os computadores têm se tornado mais potentes a cada ano, levando os usuários a esperar mais deles. Essa tendência tem também sido influenciada pelo uso da Internet, que está se expandindo, para trocar toda espécie de informação (...). Nosso apetite por softwares cada vez mais sofisticados cresce à medida que aprendemos de uma nova versão de um produto para a seguinte como o produto poderia ser aperfeiçoado. Desejamos softwares que sejam melhor adaptados às nossas necessidades, mas que, por sua vez, não torne o software somente mais complexo. Em resumo, desejamos mais.

O processo unificado ressalta a importância da comunicação com o cliente e dos métodos diretos para construir os casos de uso. Ele destaca o papel da arquitetura de *software*, sugerindo um fluxo de processo, iterativo e incremental, concebendo a sensação evolucionária, que é eficaz no desenvolvimento contemporâneo de *softwares*.

No começo dos anos 90, James Rumbaugh, Grady Booch e Ivan Jacobson iniciaram os estudos para a criação de um método unificado, o qual combinaria as características mais eficazes de cada um dos seus métodos individuais e seguiria características adicionais propostas por outros estudiosos. O resultado foi o surgimento da UML (*Unified Modeling Language*), ou seja, uma linguagem unificada de modelagem, apresentando uma notação robusta para a modelagem e desenvolvimento de sistemas orientado a objetos (OO). Em 1997, a UML tornou-se uma norma industrial para o desenvolvimento de produtos de *software* orientados a objetos.

A UML provê apoio à prática de engenharia de *software* OO. Atualmente, o Processo Unificado e a UML são muito utilizados em projetos OO. Diversos produtos podem ser produzidos como consequência da implementação da UML, que será descrita na seção a seguir.

1.2.2 Unified Modeling Language (UML)

De acordo com (JACBSON, 2000), a modelagem é a parte central de todas as atividades que levam à implantação de um bom sistema.

A UML é uma linguagem padrão para a confecção e a modelagem da estruturação de projetos de *software*. É empregada para visualização, especificação, construção e documentação de artefatos que utilizam o uso de sistemas de *software* considerado complexos.

A visualização do artefato através de sua representação UML refere-se a uma semântica definida. Isto auxilia o programador a utilizar a UML para modelar o código, possibilitando que outro programador, ou até mesmo qualquer outra ferramenta, seja capaz de interpretá-lo, evitando erros ou ambiguidades.

Em termos de especificação, a UML atende as decisões importantes no que se refere à análise, projeto e implementação, de modo a construir modelos precisos e completos de estrutura de *software*.

Permite ainda a documentação da arquitetura do sistema baseada em diagramas. Proporciona uma linguagem para a expressão de requisitos e a execução de testes. Possibilita ainda, uma linguagem para modelagem das atividades de planejamento do projeto de confecção do *software*.

A Figura 6 apresenta as fases do PU, relacionando-as com as fases genéricas já descritas anteriormente.

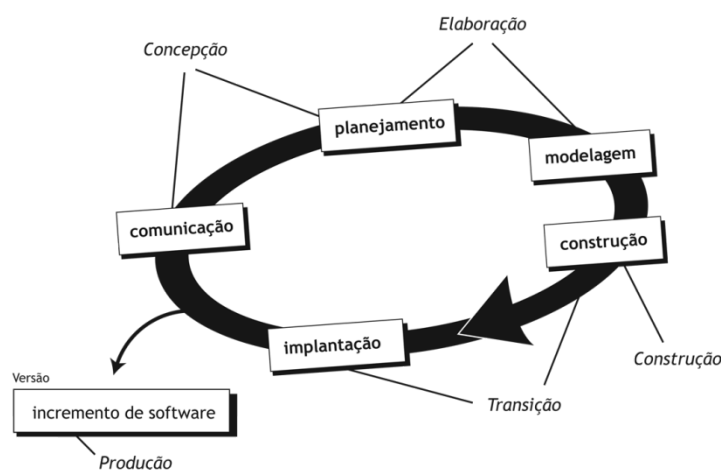


Figura 6 - Fases do PU, adaptado de (PRESSMAN, 2006).

A fase de compreensão do PU compreende as tarefas de planejamento e comunicação com o cliente. Em parceria com o cliente e os usuários finais, é feito um rascunho da arquitetura do sistema a ser desenvolvido. Neste ponto, a arquitetura refere-se a um esboço abrangendo os principais subsistemas e suas funções. Em uma fase posterior, ela será refinada e expandida em um conjunto de modelos que representarão diferentes visões do sistema. Em linhas gerais, o caso de uso delimita uma sequência de ações que são executadas por um ator, isto é, uma pessoa (funcionário), uma máquina, um sistema. O caso de uso é importante para auxiliar na tentativa de se identificar o escopo do projeto, e ainda fornecer o embasamento para se planejar o projeto.

O planejamento verifica os recursos, aferindo seus riscos e falhas. Define ainda um cronograma com prazos e metas e estabelece um embasamento para as fases que precisam ser aplicadas à medida que o incremento de *software* é desenvolvido.

A fase de elaboração inclui a comunicação com o cliente e as atividades de modelagem. Esta fase refina e expande os casos de uso que foram previamente escritos. Expande ainda a representação da arquitetura de modo a adicionar cinco visões de modelagem distintas do *software*: modelo de usos de caso, modelo de análise, modelo de projeto, modelo de implementação e o modelo de implantação. Nesta dissertação, será mostrado com detalhes o modelo de casos de uso.

A fase de construção do PU desenvolve os componentes de *software* que transformarão os casos de uso em casos operacionais para os usuários finais. Os modelos de análise e projeto que foram feitos na fase de elaboração são terminados para que a versão final

do *software* seja efetivada. O código fonte implementará todas as características e funções necessárias. Ao longo da implementação dos componentes, são realizados os testes e verificações.

A fase de transição do PU compreende os últimos estágios da atividade de construção e o início da implantação. Com isso, o *software* é testado, e relatórios são reportados com defeitos, falhas, erros e sugestões de alterações. Nesta fase são escritos os manuais de usuários, guia de soluções de suporte a problemas bem como procedimentos e manuais de instalação.

A fase de produção abrange a atividade de implantação, com a monitoração da utilização do *software* e suporte para a operação. Os relatórios de falhas são avaliados.

O fluxo de trabalho da Engenharia de *Software* é disseminado ao longo de todas as fases do PU. É importante salientar que o fluxo de trabalho identifica as atividades necessárias para se executar uma tarefa de Engenharia de *Software* e os produtos que são produzidos em consequência do término bem sucedido das tarefas.

Com isso, nas últimas décadas, os diferentes modelos de processos de *software* têm sido implementados a fim de trazer otimização e êxito para o desenvolvimento de *softwares*. Apesar de cada um destes modelos sugerirem fluxos de processos distintos, todos realizam o mesmo conjunto de atividades: comunicação, planejamento, modelagem, construção e implantação.

1.3 Redes de Computadores

A disputa acirrada pela informação tem se transformado em sinônimo de vantagens na corrida para se angariar uma melhor posição no mundo corporativo. Com isso, a procura por investimentos em Tecnologia da Informação tem se tornado cada vez mais importante e presente no cotidiano de uma organização. É neste âmbito que ocorre a disseminação das redes de computadores, uma vez que elas são responsáveis por diminuir não apenas a distância entre os equipamentos, mas ainda o tempo de resposta às requisições. É ainda em consequência da ótima relação custo-benefício que o emprego das redes de computadores vem se proliferando de forma bastante exponencial. Este aumento da notoriedade das redes de computadores acarreta, diretamente, no aumento da complexidade dos sistemas integrados de gerência. Isto ocorre porque, cada vez mais, novos dispositivos são inseridos na rede, com a

necessidade de comunicação entre eles, e com isso, a tarefa de gerenciar a rede está se tornando cada vez mais indispensável e crucial para o seu bom desempenho.

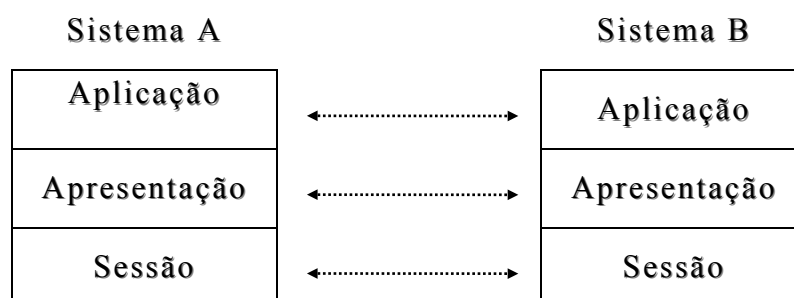
De acordo com (SAYDAM, 1996), o melhor significado de gerenciamento de rede é definido por: “*Gerenciamento de rede inclui o oferecimento, a integração e a coordenação de elementos de hardware, software e humanos, para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede e de elementos, para satisfazer às exigências operacionais, de desempenho e de qualidade de serviço em tempo real a um custo razoável*”.

1.3.1 O Modelo de Referência OSI

O tráfego de rede é provocado quando ocorre uma solicitação ou requisição na rede, e é enviado na forma de pacote de dados. Entende-se por pacote de dados a informação de um usuário transformada em um formato capaz de ser compreendido pela rede. De modo geral, todas estas informações derivam do modelo de referência do *Open Systems Interconnection* (OSI), desenvolvido pela *International Organization for Standardization* (ISO), conhecido como Modelo OSI de Sete Camadas. Este modelo é utilizado como diretriz pelos desenvolvedores de aplicações e programas de rede de computadores. Embora muitos fabricantes adaptem o a configuração deste modelo, o modelo OSI ainda é a base e referência utilizada neste desenvolvimento. A Figura 7 ilustra o modelo OSI (com exceção do meio físico).

Como de praxe, a apresentação do Modelo OSI é feita a partir da camada mais inferior na hierarquia do modelo de sete camadas, a camada física.

A pilha de protocolos recebe os dados e executa a comunicação entre cada camada, onde processa a informação recebida para comunicar-se apenas com a camada imediatamente superior e inferior a ela. Na transmissão, cada camada do Modelo OSI recebe da camada imediatamente superior os dados enviados. É nesta fase que o controle da camada é inserido, e após esta operação, a camada referida envia os dados para a camada imediatamente inferior.



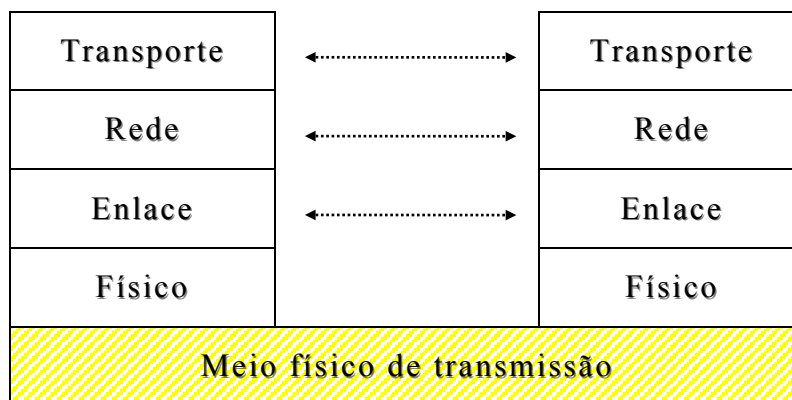


Figura 7 - Modelo OSI.

Na recepção, tal qual ocorre na transmissão, cada camada recebe os dados da camada imediatamente inferior. É nesta fase que ocorre a verificação e remoção dos controles inseridos na etapa da transmissão. Após esta operação, a camada está apta para enviar os dados para a camada superior.

Abaixo se encontra uma breve descrição da função de cada camada do Modelo OSI.

A Camada Física

A principal função da camada física é transmitir os bits por um canal de comunicação. O projeto da rede de computadores deve assegurar que o valor do bit enviado será o mesmo do recebido. Outras características importantes desta camada são a voltagem para representação dos bits (0 e 1), o método de estabelecimento da primeira conexão e do término quando ambos os lados tiverem encerrado a transmissão. O escopo do projeto deve conter cálculos de interface mecânica, elétrica e preocupar-se com o meio físico de transmissão, o qual se situa abaixo da camada física.

Figura 8 - O Modelo de Referência OSI (TANEMBAUM, 2011).

A Camada de Enlace de Dados

Esta camada é transformada um canal de transmissão bruta em uma linha que pareça correta, em termos de transmissão, ou seja, sem erros não detectados para a camada de rede. Em linhas gerais, tem como função apresentar à camada superior a filtragem dos erros da camada física. Para isto, é necessário que o transmissor divida os dados de entrada em quadros de dados de modo a transmiti-los sequencialmente. Caso este serviço seja confiável, a recepção dos dados corretos será confirmada pelo receptor, de modo a enviar um quadro de confirmação para cada recebido.

A Camada de Rede

A principal tarefa desta camada é gerenciar a operação da sub-rede, tratando o roteamento dos pacotes de origem para o destino. Em um cenário com muitos pacotes na sub-rede simultaneamente, conseqüentemente estes compartilharão a mesma rota, o que pode ocasionar um considerável gargalo na rede. O controle deste congestionamento no tráfego de dados é função da camada de rede, que por sua vez, ainda é responsável por gerenciar a qualidade do serviço, como retardo, tempo em transito e instabilidade da rede. A camada de rede precisa garantir que a utilização de protocolos diferentes não afete a transmissão de dados, ou seja, garantir a interconexão de redes heterogêneas.

A Camada de Transporte

Esta camada aceita os dados da camada acima dela, divide-os em quadros menores de acordo com a necessidade, e é responsável por enviar estas unidades à camada de rede, de modo a garantir que todos os fragmentos sejam recebidos de forma correta no lado receptor. É importante que as camadas superiores fiquem isoladas das possíveis mudanças na topologia da rede e de *hardware*.

A camada de transporte ainda especifica o tipo de serviço mais adequado à camada de sessão e aos usuários da rede. O tipo de conexão de transporte utilizado mais comum é um canal ponto a ponto livre de erros, que entrega mensagens (ou pedaços de mensagens, divididas em bytes) na ordem em que elas são enviadas. Também existe a mensagem sem a garantia respectiva à ordem de entrega. O tipo de serviço é especificado quando a conexão é estabelecida. A camada de transporte é uma camada fim a fim, uma vez que conecta a origem ao destino, onde os protocolos são trocados entre máquina origem e destino, mesmo estando separadas por diversos elementos de rede, como roteadores.

A Camada de Sessão

A camada de sessão possibilita o estabelecimento da sessão entre os processos existentes que estão em execução, mas em estações diferentes. Alguns serviços típicos desta camada são: controle de diálogo, reconhecimento de nome, verificação periódicas da transmissão para assegurar que, em caso de falha, elas continuem a partir do ponto em que estavam ao ocorrer uma falha. Outra operação importante é o gerenciamento do token, o que evita que um usuário realize a mesma operação ao mesmo tempo em ambos os lados de uma dada sessão.

A Camada de Apresentação

Nesta camada, os dados são formatados para visualização e utilização em equipamentos específicos, possibilitando a apresentação para a camada de aplicação, de modo que pode ser entendida como uma camada responsável por converter os dados na rede. Há uma preocupação com a sintaxe e a semântica das informações transmitidas. A camada de apresentação converte os dados de um formato utilizado pela camada de aplicação para um formato comum no dispositivo de envio para um formato conhecido e inteligível para a camada de aplicação, no dispositivo que receberá este formato, ou seja, a estação de recebimento.

Esta camada apresenta ainda as funções de tradução do código de caracteres, conversão de dados, compactação de dados (contribuindo para a redução de bits que são transmitidos na rede) e ainda a criptografia de dados para fins de segurança.

A Camada de Aplicação

Esta camada pode ser entendida, de maneira abstrata, como a janela para usuários e processos de aplicações que desejam acessar os recursos disponíveis na rede. É responsável pela comunicação das aplicações com a pilha de protocolos de comunicação.

Algumas das principais funções desta camada são: acesso remoto a arquivos e impressoras compartilhadas na rede, gerenciamento da rede e etc.

1.3.2 Arquitetura de Protocolos da Internet

A arquitetura de protocolos da Internet segue o modelo de rede em quatro camadas da arquitetura TCP/IP. Os protocolos pertencentes ao conjunto de protocolos TCP/IP encontram-se localizados nas três camadas superiores deste modelo. A Figura 8 apresenta a arquitetura de protocolos da Internet.

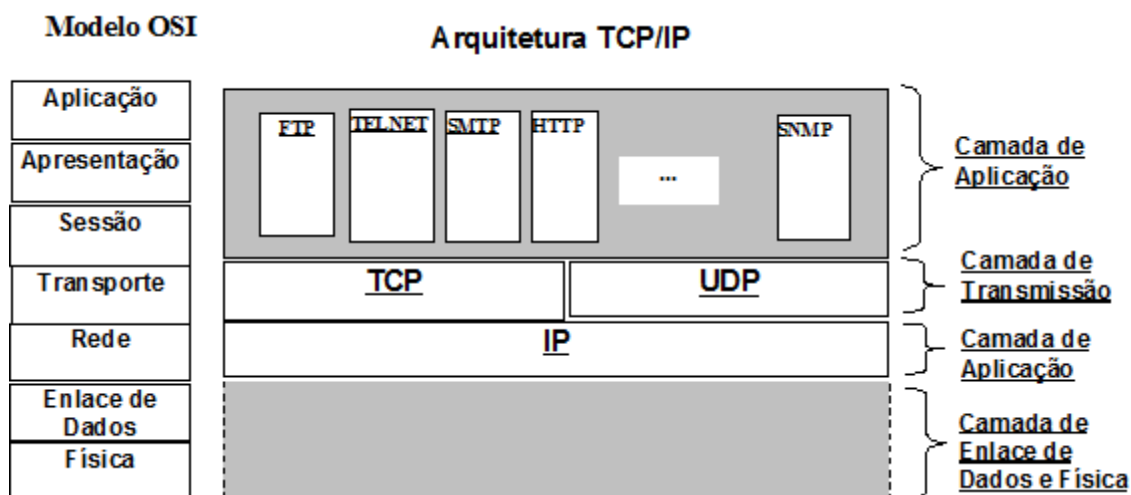


Figura 9 – Arquitetura de Protocolos da Internet (BERNARDO FILHO, 1999).

Uma breve descrição das camadas deste modelo é feita a seguir:

A Camada de Aplicação

Refere-se a um conjunto amplo de protocolos que oferecem serviços específicos em nível do usuário. Ela contém todos os protocolos de nível mais alto, como o Protocolo de Terminal Virtual (TELNET), o Protocolo de Transferência de Arquivos (FTP) e ainda o Protocolo de Correio Eletrônico (SMTP). O TELNET admite que um usuário acesse remotamente um outro terminal. O FTP permite que o usuário ou um programa mova arquivos de um terminal para o outro. O SMTP é o protocolo responsável pela especificação de como dois sistemas de correio eletrônico devem interagir (cliente e servidor).

A Camada de Transmissão

As funções da camada de transmissão podem ser executadas pelos protocolos TCP ou UDP, dependendo do tipo de serviço de conexão utilizado. O protocolo TCP (*Transmission Control Protocol*) oferece serviços de comunicação orientados à conexão, de maneira confiável. O protocolo UDP (*User Datagram Protocol*) oferece serviços do tipo datagrama, isto é, não orientados a conexão.

Camada de Enlace de Dados e Física

A arquitetura TCP/IP não impõe nenhuma restrição em relação à implementação dos níveis de enlace de dados e físico das redes que interliga. Os *backbones* da Internet são

geralmente constituídos através de padrões de redes de alta velocidade, como ATM (*Asynchronous Transfer Mode*), *Frame Relay* ou SMDS (*Switched Multimegabit Data Service*).

1.4 Segurança

A necessidade da segurança, segundo (NAKAMURA, 2003), nas décadas de 70 e 80, era a manutenção do sigilo dos dados nos negócios. Nas décadas de 80 e 90, com o início do ambiente de expansão da rede de computadores, o enfoque mudou para a integridade, no sentido de que a proteção e segurança eram feitas tendo em mente a informação e não os dados. Posteriormente, com a chegada da década de 90, a computação ganhou destaque com o crescimento acelerado das redes, e a abordagem passou a ser disponibilidade, além da proteção dos dados.

O cenário atual da Internet evidencia a importância da segurança nos sistemas computacionais. De um lado, a rede mundial se mostra indispensável a cada dia; e do outro, demonstra que é necessário muito cuidado com a proteção dos sistemas, a fim de se diminuir as perdas de dados e financeiros com incidentes de segurança.

Ainda conforme (NAKAMURA, 2003) “(...) *segurança é inversamente proporcional às funcionalidades*”. Deste modo, quanto maior o número de funcionalidades que um sistema é capaz de disponibilizar, proporcionalmente, maior será a chance de ocorrer alguma vulnerabilidade que apresente a possibilidade de ser explorada, resultando em um menor nível de segurança do ambiente e maior responsabilidade dos administradores do sistema.

Em linhas gerais, um serviço de segurança é encarregado de impedir eventos que possam comprometer a disponibilidade e a escalabilidade, a integridade ou a confidencialidade de um recurso físico ou computacional, por meio do controle, mitigação ou eliminação de ameaças, riscos e vulnerabilidades. Na implementação de um serviço de segurança de rede, podem ser usados mecanismos de controle com o intuito de prevenir, recuperar ou detectar um ataque. Os Sistemas de Detecção de Intrusão (SDI) se enquadram como mecanismos de controle de detecção de ataques. Estes sistemas têm como finalidade detectar ataques de nós maliciosos e aplicar as contra medidas adequadas a fim de manter a operacionalidade da rede. O bom funcionamento de uma rede ou sistema depende das medidas que serão tomadas após o processo de detecção e análise dos alertas emitidos pelos equipamentos de gerência. Como já citado anteriormente, no cenário das redes de

computadores, a implementação de mecanismos de segurança é uma necessidade latente. A utilização de um tipo de mecanismo de segurança não pode impactar de forma relevante os recursos disponíveis da rede. Este impacto também deve ser também alvo de estudos, além da própria verificação da eficiência do mecanismo adotado.

As invasões exploram deficiências na concepção, implementação, configuração, suporte ou no gerenciamento de sistema e serviços. Atualmente podem ser encontrados muitos estudos e técnicas que permitem a criação de mecanismos de segurança que atuem de forma automatizada, agindo e reagindo de forma rápida e eficaz no bloqueio das ações que tem como objetivo danificar o ambiente.

Apesar de o conceito mais difundido de segurança seja relacionado apenas à confidencialidade, ele é mais abrangente. De acordo com alguns autores, a questão de segurança pode ser dividida em três propriedades: confidencialidade, integridade e disponibilidade. Dependendo do cenário, um aspecto pode apresentar maior relevância do que outro. A avaliação do tipo de segurança necessária influencia diretamente na tomada de decisão sobre os projetos e técnicas. As propriedades são listadas abaixo:

- **Confidencialidade:** de acordo com (RUSSEL, 2007), uma dada informação não pode ser revelada a menos que exista uma autorização para isso. Esta propriedade é a mais conhecida por estar relacionada a diversos incidentes de segurança. Sempre que um atacante invade um sistema, ela é infringida. Está ainda associada ao sigilo das informações. Segundo a NBR ISO/IEC 17799:2001, a confidencialidade é a garantia de que a informação será acessada somente por pessoas previamente autorizadas.
- **Integridade:** um parâmetro fundamental para que um sistema seja considerado protegido e seguro é a manutenção contínua da integridade das informações registradas nele. A integridade está associada à garantia de exatidão e completeza da informação (ABNT 2001). Ainda de acordo com (DETSCH, 2005), do ponto de vista das técnicas empregadas, a verificação de integridade se assemelha à autenticação, uma vez que o emprego de um código de autenticação da mensagem também garante a integridade de forma adequada.
- **Disponibilidade:** além dos tópicos já citados, um sistema seguro necessita disponibilizar seus dados e informações para seus usuários. Um sistema que possui alta disponibilidade é capaz de permanecer em operação por muito tempo, e em caso

de eventuais problemas, sua recuperação deverá ocorrer de maneira rápida e eficaz. A disponibilidade está relacionada ao estado da informação, que pode estar disponível ou indisponível. A NBR ISO/IEC 17799: 2001 define o termo disponibilidade como sendo a garantia de que os usuários autorizados obtenham acesso à informação, e aos ativos correspondentes sempre que seja necessário. É importante diferenciar conceitos de disponibilidade. Neste contexto, disponibilidade não significa que os dados têm a obrigação de estar *online* o tempo todo, e sim que devem estar *online* sempre que o usuário autorizado os requisitar.

De acordo ainda com (RUSSEL, 2007), são três elementos básicos que devem coexistir na operação da segurança: vulnerabilidades, ameaças e mecanismos de segurança. Entende-se por vulnerabilidade o ponto de falha onde o ambiente está apto a ser atacado. O conceito de ameaça é um pouco distinto de vulnerabilidade. Entende-se por ameaça o indício ou eminência de perigo, as quais possuem a capacidade de comprometer um sistema computacional ou a própria rede. Esta ameaça pode ter como agente causador uma pessoa (*hacker*, espião, ou qualquer pessoa mal intencionada), um evento (falha em um dispositivo ou equipamento) ou um desastre natural (fogo, enchente, explosão). É importante salientar que as ameaças só afetam um cenário se houver alguma vulnerabilidade permitindo que isto ocorra. Os mecanismos de segurança atuam como um instrumento para a proteção dos sistemas e da própria rede, de modo que auxiliem no processo de dificultar as ações das ameaças sobre as próprias vulnerabilidades.

1.4.1 Vulnerabilidades

Entende-se por vulnerabilidade as características inerentes ou falhas do sistema, que podem prejudicar o seu funcionamento considerado normal. Partindo-se da premissa que, por definição, não é possível existir um ambiente totalmente seguro, todos os sistemas de computadores são vulneráveis a ataques. Políticas de segurança e mecanismos com ferramentas específicas auxiliam na redução das estatísticas de uma tentativa de ataque. Todavia, há outros tipos de ataque que merecem atenção e precisam ser evitadas.

As vulnerabilidades podem ser classificadas de acordo com o tipo de risco que apresentam. Na lista abaixo, são indicadas as vulnerabilidades mais comuns:

- Vulnerabilidades físicas: é o tipo de vulnerabilidade presente onde o acesso às instalações físicas do sistema é vulnerável. Pessoas mal intencionadas podem roubar

equipamentos, como mídias e outros dispositivos de armazenamento de informações até danificarem inteiramente um computador. A presença de vigilantes, câmeras de monitoração e sistemas de leitura biométrica amenizam estes riscos.

- Vulnerabilidades naturais: computadores são vulneráveis a desastres naturais e ameaças ambientais, tais como: incêndios, enchentes, terremotos, raios, chuvas ácidas, etc, que podem comprometer o sistema.
- Vulnerabilidades de *hardware* e *software*: alguns tipos de falhas de *hardware* e *software* podem afetar a segurança de todo o sistema. Para exemplificar, é possível citar a aquisição de acesso à partes reservadas de memória por um processo não autorizado. Caso a segurança da memória não funcione como o esperado, um outro processo pode assumir um privilégio que inicialmente, não seria autorizado caso o *hardware* se comportasse da maneira prevista. É importante mencionar que as falhas de *software* também são capazes de possibilitar lacunas e aberturas para que um sistema seja considerado vulnerável. Com isso, a facilidade na exploração deste tipo de vulnerabilidade, as falhas de *softwares* são constantemente exploradas e, conseqüentemente, novas vulnerabilidades diárias são lançadas diariamente.
- Vulnerabilidades de comunicação: computadores conectados na rede apresentam o fato de segurança ainda mais comprometido. Mensagens trocadas podem ser facilmente interceptadas se não possuírem mecanismos de segurança nem criptografias adequadas.
- Vulnerabilidades humanas: usuários e também os próprios administradores de computadores representam uma importante fatia em vulnerabilidades. Seres humanos são, na maioria das vezes, imprevisíveis e capazes de fazer qualquer coisa, conscientemente ou não, que coloquem o sistema em condições de risco.

1.4.2 Ameaças

Entende-se por ameaças os possíveis perigos à segurança de um ambiente. Em caso de haver uma ameaça no sistema, ela pode se aproveitar das vulnerabilidades para adulterar a confidencialidade, integridade ou disponibilidade de dados. De acordo com (RUSSEL, 2007), as ameaças podem ser categorizadas em três modalidades:

- Naturais e físicas: incêndio, enchente, terremoto, desabamento, furacão, etc. Este tipo de ameaça pode ser prevenida, porém não é possível saber quando acontecerá.
- Não intencionais: como o nome diz, são ameaças onde não houve a intenção por parte das pessoas envolvidas. Descuido, falta de informação e atenção são os responsáveis por este tipo de ocorrência, capazes de comprometer e danificar o sistema ou a própria rede. A falta de criptografia na transmissão dos dados também é um fator que compromete a segurança do sistema.
- Intencionais: estes tipos de ameaças são oriundos de pessoas mal intencionadas, que deliberadamente desejam se apoderar de dados, serviços ou recursos do sistema, de modo a obter algum ganho com esta operação.

1.4.3 Mecanismos de Segurança

Mecanismos de segurança são capazes de identificar e evitar que as ameaças sejam capazes de afetar os sistemas. Sua relevância é proporcional à importância dos sistemas que protegem. Os tipos de ataque mais conhecidos são citados a seguir.

Um ataque de rede ou segurança, ou ainda um incidente de segurança é definido como uma ameaça, invasão, negação de serviço ou outro ataque a uma dada infraestrutura de rede que analisará sua rede a fim de obter informações para, eventualmente, causar falhas na mesma. Em muitos casos, o atacante pode não estar interessado apenas em explorar as aplicações de *software*, como também obter acesso não autorizado a dispositivos de rede. De forma geral, dispositivos de rede sem monitoramento são a principal fonte de vazamento de informações nas organizações. Na maioria dos cenários existentes atualmente, cada dispositivo de rede trata uma mensagem de *e-mail*, uma solicitação de página *web*, uma solicitação de *login* do usuário e uma transmissão de arquivos. Em algumas configurações ainda, serviços de telefonia e mensagens de voz também são tratados por dispositivos de rede. Se o atacante é capaz de "possuir" seus dispositivos de rede, então há a possibilidade de "possuir" toda a sua rede.

Nesta dissertação serão apresentados os principais tipos de ataques:

1. Spoofing

Qualquer dispositivo conectado à internet envia datagramas IP para a rede. Esses pacotes de dados identificam o endereço IP do remetente, bem como dados da camada de aplicação. Se o atacante obtém controle sobre o software rodando em um dispositivo de rede, é possível modificar os protocolos do dispositivo para configurar um endereço IP aleatório no campo de endereço de origem do pacote de dados. Isto é conhecido como falsificação de IP, o que acarreta que qualquer informação pode vir de qualquer fonte. Com um endereço IP de origem falsificado em um datagrama, é difícil encontrar o host que realmente enviou o datagrama.

Uma medida para evitar para spoofing é a filtragem dos dados que entram na rede. Roteadores fazem isso: executam a filtragem e verificam o endereço IP de datagramas de entrada e determinam se os endereços de origem que são. Se o endereço da fonte não está na faixa válida de endereços IPs conhecidos, os pacotes serão descartados.

2. *Sniffing*

Sniff, do inglês, significa farejar. Trata-se da interceptação de pacotes de dados que atravessam a rede. Um programa *sniffer* funciona na camada Ethernet em combinação com placas de interface de rede (NIC) para capturar todo o tráfego. Além disso, se qualquer uma das placas de rede Ethernet está em modo promíscuo, o programa *sniffer* vai coletar todos os pacotes de comunicação que flutuam perto do local do host da internet. Um *sniffer* colocado em qualquer dispositivo *backbone*, *link* ou rede de inter-rede ponto de agregação será, portanto, capaz de monitorar um monte de tráfego. A maioria dos *sniffers* são passivos e escutam todos os quadros da camada de enlace de dados.

A chave para a detecção deste tipo de ataque é detectar interfaces de rede que estão em execução no modo promíscuo. O *sniffing* pode ser detectado de dois modos:

- a. Baseada em máquinas: existem comandos de *software* que podem ser executados em máquinas para detectar se a placa de rede está funcionando em modo promíscuo.
- b. Baseada em rede: soluções tendem a verificar a presença de processos em execução e arquivos de *log*. No entanto, os intrusos sofisticados quase sempre escondem seus rastros, disfarçando o processo e apagando os arquivos de *log*.

A melhor medida preventiva contra o *sniffing* é a criptografia.

3. *Mapping*

Antes de atacar a rede, os intrusos precisam saber algumas informações como o endereço IP das máquinas na rede, os sistemas operacionais que utilizam e os serviços que eles

oferecem. Com essa informação, os ataques podem ser mais direcionados e são menos propensos a causar alarme. Esse processo de coleta de informações é conhecido como mapeamento.

Em alguns ambientes, as comunicações de rede ocorrem em um formato "*clear text*" inseguro, o que permite que um atacante obtém acesso a caminhos de dados em sua rede para "ouvir" ou interpretar o tráfego. Quando um atacante está espionando suas comunicações, é referido como *sniffing*. A capacidade de um espião para monitorar a rede é geralmente o maior problema de segurança que os administradores enfrentam.

Medidas de proteção são serviços de criptografia. Caso contrário, os dados podem ser lidos por outras pessoas, uma vez que atravessa a rede.

4. *Man in the middle*

Esta é uma técnica que se aproveita de uma vulnerabilidade na pilha de protocolos TCP / IP, e a forma como os cabeçalhos são construídos. Ocorre quando alguém entre o usuário e o outro usuário com quem está se comunicando está ativamente monitorando, capturando e controlando a comunicação de forma transparente.

Os ataques *Man in the middle* são entendidos como alguém assumindo sua identidade, a fim de ler sua mensagem. A pessoa do outro lado pode acreditar que é você, uma vez que o atacante pode estar respondendo ativamente como você, para obter mais informações.

5. *Trojans*

Estes são os programas que se assemelham com *software* comum, mas, na verdade, executam ações não intencionais ou maliciosas em segundo plano quando lançados. É o tipo de ataque mais comum utilizado. O número de técnicas de *trojans* é infinita. A única proteção é usar uma soma de verificação criptográfica ou procedimentos de assinatura digital binária.

6. Negação de Serviço (DoS – *Denial of Service*)

Um ataque de negação de serviço é um tipo especial de ataque de Internet destinado a grandes *sites*. É um tipo de ataque que é projetado para derrubar a rede, inundando-a com tráfego inútil. Negação de serviço pode resultar em um sistema, tal como um servidor web, que seja inundado com pedidos ilegítimos, tornando-se impossível atender a solicitações reais.

Um ataque DoS pode ser cometido de várias maneiras. Existem três tipos básicos de ataque.

- Consumo de recursos computacionais, como largura de banda, espaço em disco e tempo de CPU.
- Rompimento de informações de configuração, tais como informações de roteamento.
- Rompimento de componentes de rede físicas.

As consequências de um ataque de negação de serviço são as seguintes:

- Desempenho da rede extraordinariamente lento.
- Indisponibilidade de um site particular.
- Incapacidade de acessar qualquer site da web.
- Aumento exponencial na quantidade de spam recebido na conta alvo.

1.4.4 Técnicas de Invasão e Descoberta de Informações

De acordo com alguns autores (MCCLURE, 2000) e (HATCH, 2002), os ataques podem ser divididos em três etapas distintas, a saber: mapeamento de vulnerabilidades, invasão e pós-invasão. Ultimamente uma prática bastante comum é a verificação das vulnerabilidades existentes em sistemas conectados a Internet.

Os métodos e mecanismos indicados para a proteção de sistemas computacionais conectados na rede variam de acordo com o foco dado. Em (BENSO, 2003) são indicadas seis abordagens diferentes para a prevenção às intrusões:

- Prevenção: prever ou coibir uma invasão. *Firewalls* são um exemplo desta abordagem, onde, pelo filtro dos pacotes de dados, é possível prevenir que algumas tentativas de intrusão consigam afetar os computadores protegidos pelo dispositivo.
- Preempção: é a ação contra o atacante antes que ele invada o sistema.
- Persuasão: é a tática de persuadir o atacante a não realizar o ataque ou mesmo suspender um possível ataque que já está ocorrendo. Em alguns cenários, diante de

ataques distribuídos de negação de serviço, esta persuasão pode ocorrer através de quantias monetárias enviadas aos atacantes.

- Ilusão: iludir o atacante, de modo que pense que a invasão tenha sido bem sucedida. Esta técnica é utilizada nos *honeypots*.
- Detecção: consiste em identificar as tentativas de intrusão do sistema e ativar os mecanismos apropriados. Nesta técnica encontram-se os SDI.

1.4.5 Sistema de Detecção de Intrusos

Segundo (BARBOSA, 2000) segue que: “*Os Sistemas de Detecção de Intrusos (SDI) são responsáveis por identificar, relatar e combater atividades maliciosas provenientes tanto de elementos externos quanto de elementos internos ao sistema*”. No ano de 1986, (DENNING, 1986) propôs pela primeira vez um modelo para detectar tentativas de invasão, penetração e outros tipos de abusos durante a utilização do computador conectado à rede. Desde então, inúmeras sugestões foram propostas para suprir as necessidades, carências e deficiências dos IDS existentes, como o NetRanger (NetRanger 2.2.1, 2012) e NID (Lawrence Livermore National Laboratory, 2012). Posteriormente, surgiram o BRO (PAXSON, 1998) e NetStat (VIGNA, 2007), baseados na verificação de assinaturas digitais. No entanto, estes sistemas carecem de atualização cada vez que um novo ataque é descoberto e divulgado, dificultando as tarefas do administrador da rede, uma vez que, além de observar os relatórios dos IDS, é necessário estar em alerta para as atualizações de banco de dados.

Uma intrusão pode ser definida como o uso não autorizado, mau uso ou ainda um abuso feito por usuários conhecidos ou invasores desconhecidos em um determinado sistema computacional (TILLAPART, 2002). Outra definição encontrada que possui o alicerce nos conceitos de segurança: *uma intrusão é um conjunto de ações que visa comprometer a integridade, confidencialidade ou a disponibilidade dos dados ou do sistema* (FERREIRA, 2003).

Os estudos realizados sobre os sistemas de detecção de intrusos já datam de 30 anos. Em 1980, James Anderson publicou um dos primeiros trabalhos sobre o tema, denominado *Computer Security Threat Monitoring and Surveillance*. A detecção de intrusão é o ato de identificar indivíduos que utilizam um sistema computacional sem autorização, e também usuários legítimos, que estão autenticados e credenciados no sistema, mas que abusam dos

privilégios concedidos pelo administrador da rede (TILLAPART, 2002). Isto é uma referência aos riscos dos invasores e usuários mal intencionados internamente. Os usuários conhecidos do sistema, que na maioria das vezes estão alocados dentro da própria estrutura de rede, mas que por alguma razão decidem colocar em risco a segurança dos sistemas podem gerar grandes e graves incidentes. É importante ainda apresentar outra definição focada na análise de políticas de segurança adotadas nas instituições: a detecção de intrusão é o processo de, inteligentemente, monitorar eventos ocorrendo em um sistema computacional ou na rede, e diagnosticá-los por meio das violações das políticas de segurança (IDRIS, 2006). É importante observar que esta segunda definição se refere à utilização de sistemas de inteligência computacional ao mencionar o termo “inteligentemente”, o qual se aplica a esta dissertação.

No cenário atual, juntamente com os *firewalls*, os sistemas de detecção de intrusão têm-se tornado um componente necessário na maioria dos sistemas de segurança de rede. O seu principal objetivo é identificar potenciais vulnerabilidades, ataques ou riscos nas políticas de segurança (MILLER, 2003). Antes da implantação de um SDI, é necessário que sejam verificadas as razões técnicas para sua implantação (BACE, 2011), listadas a seguir:

- Prevenção de comportamentos maliciosos e punição dos envolvidos nas violações cometidas aos sistemas que são monitorados;
- Detectar violações na segurança da rede e dos dispositivos que não sejam verificados por outros mecanismos;
- Detectar e tomar ações iniciais contra os ataques que estão se iniciando;
- Documentar e especificar em relatórios a existência de ameaças aos sistemas;
- Fornecer informações frequentes e comuns sobre intrusos ocorridas, permitindo uma melhoria no diagnóstico, recuperação e correção dos fatores causadores dos incidentes.

As motivações para um ataque são diversas, variando de acordo com o atacante. A motivação pode ser pelo aprendizado, dinheiro, fama, necessidades psicológicas, vingança,

espionagem industrial ou curiosidade (NAKAMURA, 2003). Identificar uma intrusão não é uma tarefa fácil. As redes atualmente existentes são bastante complexas, interligando sistemas operacionais distintos com múltiplos serviços e operando com uma diversidade bastante abrangente de protocolos. A velocidade da necessidade do surgimento de novos protocolos e aplicações dificulta ainda mais a tarefa de detectar intrusões (SCHNEIER, 2002).

A precisão na distinção entre eventos classificados como normais e os indicativos de uma efetiva intrusão (KABIRI, 2005). É essencial que o SDI seja capaz de não gerar falso-positivos, isto é, não gerar alarmes para situações consideradas normais e já conhecidas. E ainda não deixar de enviar alarmes para as reais situações de intrusão, os chamados falso-positivos. Esta é a razão pela qual, na maioria dos cenários, os SDI são encontrados e configurados para reagir a ataques apenas se houver intervenção e operação de especialistas humanos. Caso houvesse uma confiança mais significativa na tomada de decisão dos sistemas de detecção, a automação das respostas, como o bloqueio, suspensão e negação de serviços pelo *firewall*, por exemplo, seria mais bem aceita e confiável. Este fato gera receios acerca das consequências de que as reações dos SDIs sejam tão nocivas quanto às ações dos próprios ataques.

Assim sendo, muitos SDI atuam como ferramentas auxiliares ao trabalho dos seres humanos, por não apresentarem um nível significativo de confiança a ponto de possuírem uma operação com resposta automatizada. Isto ocorre devido à incapacidade em responder a ataques desconhecidos ou àqueles conhecidos, mas cujo padrão de comportamento sofreu alterações ao longo dos ataques. A proposta desta dissertação é justamente esta, auxiliar na automação das respostas utilizando lógica nebulosa a fim de não ser mais necessária a dependência de um especialista.

1.5 Computação em Nuvem

O emprego da nomenclatura Computação em Nuvem (do inglês, *Cloud Computing*), vem se tornando cada vez mais frequente, com a promessa de se tornar um paradigma que transformará o modo atual de desenvolvimento e comercialização dos produtos de *softwares*. Por se referir a um termo muito recente, apresenta ainda muitas dúvidas e divergências acerca do seu conceito. Apesar de ainda não haver uma definição exata e consensual acerca do

conceito da Computação em Nuvem, este termo possui uma definição interessante e bastante abrangente para este paradigma, podendo ser utilizada como referência para esta dissertação.

Segundo (FOSTER, 2002), “*Computação em nuvem é um paradigma de computação em larga escala que possui foco em proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da Internet*”.

A Computação em Nuvem tem se transformado em uma das palavras chaves da Tecnologia da Informação. A nomenclatura nuvem é uma metáfora para a Internet ou infraestrutura de comunicação entre os componentes de sua arquitetura baseada em uma abstração que oculta a complexidade da infraestrutura. Cada parte desta infraestrutura é fornecida como um serviço e, estes são normalmente alocados em centros de dados, utilizando *hardware* compartilhado para computação e armazenamento (BUYA, 2009).

A solução da computação em nuvem apresenta custo financeiro reduzido, uma vez que para os usuários utilizarem os serviços fornecidos pela nuvem, é necessário apenas uma máquina configurada com um sistema operacional (que pode ser otimizado com configurações mínimas), um navegador (do inglês, *browser*) e placa de rede (cabada ou *wireless*, ou seja, sem fio) para o acesso à internet. Todos os recursos computacionais estão disponíveis na nuvem, sendo que os usuários não precisam se preocupar com recursos computacionais robustos, acarretando na facilidade de obtenção de baixo custo no valor das máquinas. A computação em nuvem permite um alto nível de abstração e customização, já que todo e qualquer *hardware* pode ser utilizado para realizar alguma tarefa que seja adequada ao seu poder de processamento.

Novos recursos de *hardware* e demais dispositivos podem ser adicionados com o intuito de aumentar o poder de processamento e cooperar com os recursos já existentes.

1.5.1 Modelos de Implantação na Nuvem

Os quatro principais modelos de implantação que podem ser aplicados à Computação em Nuvem são (MELL, 2009):

1. Nuvem privada: a infraestrutura é implementada para atender as necessidades de uma organização, a qual pode ser gerenciada pela organização ou por um terceiro. Em relação a sua implementação, pode ser local ou remota.

2. Nuvem baseada em comunidade: é compartilhada por várias organizações que possuem interesses afins como por exemplo, as práticas de segurança, premissas, políticas, etc. Pode ser gerenciada pelas organizações participantes da comunidade ou também por um terceiro. Sua implementação é feita localmente ou remotamente.
3. Nuvem pública: a infraestrutura é feita para o público em geral, podendo pertencer a alguma organização que vende serviços de computação.
4. Nuvem híbrida: é um arranjo entre dois ou mais modelos de nuvens, por exemplo, privado e público. O diferencial neste modelo consiste no fato de serem conectados por alguma tecnologia específica, a qual pode ser padronizada, aberta ou até mesmo proprietária.

1.5.2 Segurança na Nuvem

Por muito tempo, *datacenters* e empresas foram construídos para garantir que um usuário que não tivesse a necessidade de ter contato físico com computadores, servidores e discos. Este pensamento tem como premissa uma regra de segurança sólida que afirma que se um indivíduo tiver a chance de tocar fisicamente um dispositivo, este pode ser mais facilmente corrompido ou danificado. Para muitos estudiosos e profissionais de TI, o pensamento de hospedar aplicações na nuvem é muito preocupante. Quando as questões de segurança na nuvem são colocadas à prova, é necessário pensar em termos de dois ou três tipos de ameaças. Primeiramente, é preciso listar as ameaças que correspondem às ameaças mais comuns para soluções baseadas em nuvem e soluções baseadas em aplicações que não estejam na nuvem. A segunda lista deve se concentrar em ameaças específicas para aplicações que rodem na nuvem.

Vantagens de Segurança na Nuvem

Como os provedores de solução baseada em computação em nuvem espalham e diluem os custos através de vários clientes, significando que a maioria possui mais dinheiro disponível para investir em soluções diferentes, dentre elas, questões de segurança. A lista abaixo ilustra alguns benefícios que as empresas de soluções em nuvem devem respeitar em relação à segurança (JAMSA, 2012):

- Implementação imediata de pacotes de correção (*patches*) nos produtos de *softwares* instalados

Alguns pacotes de correção contêm conformidades específicas de segurança. Várias empresas de soluções baseadas em nuvem possuem uma equipe disponível para distribuir e instalar

estas correções. Deste modo, sistemas baseados em sistemas na nuvem podem se apresentar vulneráveis por um curto período de tempo até que um novo pacote de correção for lançado, uma vez que logo será instalado.

- Extensão do alcance das relações humanas

Devido a sua força financeira, os provedores de soluções baseadas na nuvem são aptos a reconhecer empreendedores em potencial que administrarão os sistemas.

- Redundância de *hardware* e *software*

A maioria das soluções baseadas na nuvem possui redundância de recursos de *hardware* e *software* que podem ser rapidamente distribuídos em uma situação de emergência.

- Oportunidade de respostas a incidentes

Dentro de um *datacenter*, pessoas chave, muitas vezes, executam múltiplas tarefas. Um especialista de segurança da empresa também pode ser o administrador da empresa. Como resultado, em muitas vezes pode ocorrer atrasos entre o início de um incidente de segurança e a sua identificação, o que pode gerar um resultado catastrófico. Um provedor de solução baseada na nuvem, em contraste, provavelmente tem especialistas para sistemas de monitoramento de intrusos, porcentagem de utilização do sistema e outros recursos. Deste modo, caso ocorra um incidente de segurança, o fornecedor da solução baseada em nuvem é provável que seja mais reativo.

Há ainda desvantagens em hospedar aplicações e dados na nuvem, como apresentada seguir.

Desvantagens de Segurança na Nuvem

- País ou jurisdição

Nem sempre é uma tarefa simples e evidente o local de hospedagem dos recursos baseados na nuvem. Caso a nuvem hospede seus recursos em um país remoto, por exemplo, este necessariamente precisa se preocupar com as leis do governo do país. Se os recursos da

nuvem residem em vários países, é notório que as questões como a jurisdição podem ser preocupantes em relação às questões legais.

- Riscos de múltiplos inquilinos

Algumas soluções baseadas em nuvem utilizam-se da filosofia de múltiplos inquilinos, o que significa que dois ou mais clientes podem usar os mesmos recursos. Por exemplo, um banco de dados. Como resultado deste compartilhamento de recursos, um erro na aplicação pode expor os dados de uma empresa para outra, gerando um grave incidente de segurança. Analogamente, caso um dispositivo seja compartilhado, restos de dados de outra empresa podem ser expostos à outra.

- Informantes maliciosos

Como toda empresa, as empresas baseadas em nuvem podem ser vítimas de má intenção dos funcionários também.

- Bloqueio de operadora

Dependendo de como um provedor de soluções baseado em nuvem armazena os dados de uma empresa, pode se tornar difícil para a empresa mudar de operadora, para o caso de violação de acordo de nível de serviço ou outro problema.

- Riscos de falhas

Empresas que dependem dos serviços da nuvem precisam estar cientes do risco existente do provedor vir a falhar. Algumas empresas solicitam o código fonte das aplicações, para que em caso de dano, a empresa possa obter acesso ao código e hospedar novamente a solução, precavendo-se de eventuais indisponibilidades do sistema.

1.5.3 Modelos de Serviços

Diferentes modelos de serviços atendem a diferentes tipos de requisitos. A Figura 9 lista os três principais tipos de serviços oferecidos pela computação em nuvem.

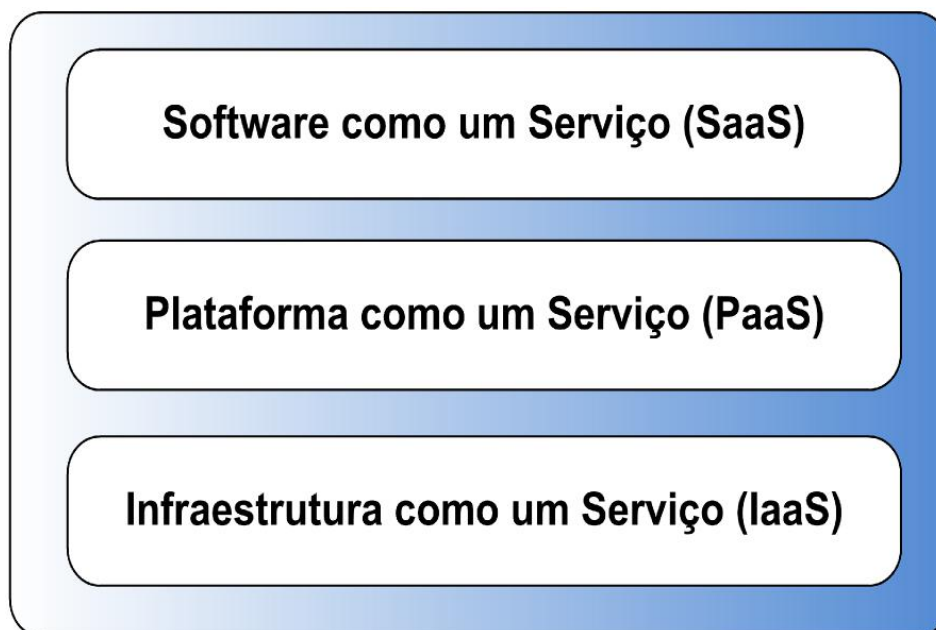


Figura 10 - Modelos de Serviços (SOUSA, 2010)

Software como um Serviço (SaaS)

Como já diz o nome, este modelo de Computação em Nuvem, o SaaS proporciona sistemas de *software* com propósitos específicos que estão disponíveis para os usuários através da utilização de Internet. Os sistemas de *software* são possíveis de serem acessados a partir de vários dispositivos do usuário por meio de uma interface *thin client* (ou seja, um cliente configurado com poucos recursos no modelo cliente-servidor), como um navegador Web. No modelo SaaS, o usuário não possui privilégios ou controles administrativos sobre a infraestrutura subjacente, tampouco, servidores, sistemas operacionais, armazenamento ou mesmo as características individuais da aplicação. A exceção é restrita às configurações específicas. Isto permite que os desenvolvedores se concentrem em inovação e não na infraestrutura, induzindo ao desenvolvimento exponencial e otimização de sistemas de *software*. O fato de o *software* estar disponível na rede permite que este possa ser acessado pelos usuários de qualquer lugar, dispositivo e momento, possibilitando uma maior mobilidade e facilidade na integração entre unidades de uma mesma empresa ou outros serviços integrados ao *software*. Novos recursos são capazes de serem integrados automaticamente aos sistemas de *software* de forma transparente para os usuários, possibilitando uma evolução e atualização dos sistemas. O SaaS reduz os custos, pois é

dispensada a aquisição de licenças de sistemas de produtos de *software*. Como exemplos de SaaS são destacáveis os serviços de *Customer Relationship Management* (CRM) da Salesforce (SALESFORCE, 2010) e o *Google Docs* (CIURANA, 2009).

Plataforma como um Serviço (PaaS)

A finalidade deste modelo de computação em nuvem é facilitar o desenvolvimento de aplicações destinadas aos usuários finais do sistema, desenvolvendo uma plataforma que dinamiza esse processo, conferindo agilidade e praticidade ao mesmo. O PaaS oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações desenvolvidas na nuvem. O usuário não possui controle e privilégios administrativos acerca da infraestrutura. Além disso, fornece ainda sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de produtos de *softwares*, já que possui ferramentas de desenvolvimento e colaboração entre desenvolvedores. Este modelo permite aos usuários utilizarem serviços de terceiros, aumentando o uso do escopo de projeto de suporte onde os usuários se inscrevem para solicitações de serviços de TI (abertura de *tickets* ou chamados) ou ainda para resoluções de problemas pela Internet. Esta característica permite o melhoramento do gerenciamento do trabalho e de governança das responsabilidades das equipes de TI das empresas. Como exemplos, pode-se citar *Google App Engine* (CIURANA, 2009) e Aneka (VECCHIOLA, 2010).

Infraestrutura como um Serviço (IaaS)

Por último, este modelo, providencia toda a infraestrutura necessária para a PaaS e o SaaS. A maior finalidade do IaaS é justamente prover facilidade e acessibilidade ao fornecimento de recursos, como por exemplo, servidores, rede de computadores, armazenamento e outros recursos de computação fundamentais para desenvolver um ambiente sob demanda. Apresenta características típicas, tais como uma interface única para administração da infraestrutura, *Application Programming Interface* (API) para interação com *hosts*, *switches* e outros dispositivos de rede. Apresenta ainda, suporte para a adição de novos equipamentos de forma simples e transparente. Na maioria dos casos, o usuário não administra ou controla a infraestrutura da nuvem, no entanto, possui controle sobre os sistemas operacionais, armazenamento e aplicativos implantados, e, se necessário, seleciona componentes para integrar à rede, conferindo maior segurança, como um *firewall*.

1.5.4 Comentários

Neste capítulo foram apresentados, de maneira sintética, os conceitos pertinentes aos assuntos abordados neste estudo, tais como: os princípios da lógica nebulosa, bem como processos de desenvolvimento de *software*, questões relativas à rede de computadores, segurança e Computação em Nuvem.

No próximo capítulo será apresentado o estado da arte para alguns destes tópicos.

2 TRABALHOS RELACIONADOS

2.1 Introdução

Neste capítulo serão descritos os trabalhos relevantes para o tema proposto. A seção 2.2 apresenta um trabalho relacionado com a Computação em Nuvem, onde é descrita uma arquitetura capaz de gerenciar identidades em nuvens híbridas. Na Seção 2.3, são apresentados três trabalhos relevantes que utilizaram um SDI. A Seção 2.4 apresenta um trabalho relacionado à Segurança de Rede. A Seção 2.5 exibe um trabalho relacionado à Gestão de Projetos utilizando a Lógica Nebulosa.

2.2 Computação em Nuvem

Em “Uma Arquitetura para Gerência de Identidades em Nuvens Híbridas” (FELICIANO, 2011) é relatada uma arquitetura para realizar a gerência de identidades em nuvens híbridas. Essa arquitetura permite que outras nuvens privadas se associem à nuvem pública da plataforma REALcloud. A implementação dessa arquitetura utilizou a infraestrutura do *middleware* OpenAM (OpenAM, 2012). Entende-se por *middleware* o programa que faz a mediação entre o *software* e as demais aplicações. O *middleware* OpenAM foi estendido para comportar serviços de autorização à rede interna de recursos. O aspecto de segurança tratado por esse trabalho foi a questão da autenticação das identidades participantes da nuvem.

No atual cenário da Internet é bastante usual que a maioria dos usuários possua várias credenciais (contas), o que significa diferentes regras e permissões de acesso em cada um dos sites acessados. Isto significa inserir um considerável nível de dificuldade no que é conhecido como experiência de navegação do usuário, dado que será necessária a utilização de várias contas nos diferentes sites. Com o advento da computação em nuvem, esta preocupação da gerência das contas passou a ser dos administradores do sistema. Outra particularidade da nuvem é sua natureza colaborativa das aplicações.

Esta questão levanta os problemas de segurança, já que a nuvem precisa verificar questões como autenticação e autorização de identidades muitas vezes vindas das demais aplicações parceiras.

Para que a cooperação citada acima transcorra sem problemas, é necessário que as entidades parceiras definam políticas para o compartilhamento de recursos junto aos outros domínios. O mecanismo de autenticação utilizado em muitos casos é o *Single Sign-On* (SSO), de modo que o usuário consiga acessar recursos de outros domínios.

2.3 Sistemas de Detecção de Intrusos

Em “Sistemas de Detecção de Intrusão com Técnicas de Inteligência Artificial” (SILVA, 2011) identifica-se a melhoria das taxas de acerto dos Sistemas de Detecção de Intrusão utilizando algumas técnicas de Inteligência Artificial. Foi implementada uma Rede Neural Artificial, que teve suas características alteradas através de Algoritmos Genéticos para obter melhores taxas de acerto sobre conexões normais e anormais de uma rede de computadores. Posteriormente, foi desenvolvido um Sistema Nebuloso, utilizado em junção com as Redes Neurais Artificiais, para formar um Sistema Híbrido Inteligente. As taxas de acerto para detecção de invasões e de tráfego normal apresentaram, em todos os métodos de IA, taxas de acerto maiores que o sistema inicial utilizado. Os resultados encontrados quanto à classificação correta dos ataques e do tráfego normal através das Redes Neurais Artificiais aumentou em até 17,6% com a utilização do Algoritmo Genético. Já o Sistema Nebuloso implementado apresentou um ganho modesto, da ordem de 5% na taxa de acerto de ataques e tráfego normal. Porém, com a junção das Redes Neurais ao Sistema Nebuloso, formando o sistema Neuro-Fuzzy, obteve-se um ganho nas taxas de acerto da ordem de 30% em relação ao trabalho original implementado.

Em *A Subjective Trust Management Model with Multiple Decision Factors for MANET based on AHP and Fuzzy Logic Rules* (XIA, 2011) é tratada a avaliação da confiança de nós que ingressam em uma rede local sem fio. Uma rede ad hoc móvel (MANET) é um sistema auto-organizável compreendido por múltiplos nós sem fio. Devido à abertura da topologia de rede e a ausência de uma administração centralizada, MANET é vulnerável a ataques oriundos de nós maliciosos.

O valor do nó é avaliado, fazendo com que o modelo se apresente mais estável, adaptativo e robusto, o que conseqüentemente aumenta a segurança da rede e seu desempenho. A simulação atestou a análise da eficácia da gestão de confiança do modelo, o qual funciona como uma interface intuitiva e eficaz no que se refere à ferramenta de análise,

avaliação e derivação, podendo fornecer apoio eficaz para as decisões de confiança e contra-ataques.

Em Implementação de um IDS utilizando SNMP e Lógica Difusa (VIRTI, 2007) é apresentado um estudo da segurança em redes de computadores através da implementação de um sistema detector de intrusos, embasado na captura de informações através da utilização do protocolo SNMP. O estudo objetiva alcançar uma diminuição no número de falsos positivo e negativo, questão característica à maioria dos IDS. Para tal, o autor utilizou lógica nebulosa para construir um sistema detector de intrusos com o auxílio da inferência de especialistas, administradores de rede, para juntos, aperfeiçoarem a segurança da rede. Os resultados obtidos na monitoração de uma rede de produção, foram superiores à expectativa, melhorando a segurança obtida com a utilização do IDS, propiciando sua adoção como mecanismos complementar a segurança de redes.

2.4 Segurança de Rede

Em “Segurança Computacional” (DUMONT, 2006) é apresentada uma reflexão sobre os perigos e riscos aos quais uma rede de computadores está sujeita. Apresenta ainda as ações e tomada de decisões que podem ser executadas para prover segurança nos servidores Linux em ambientes de redes corporativas e cooperativas. Segundo (DUMONT, 2006), o ambiente cooperativo é definido como um ambiente empresarial heterogêneo, que envolve matrizes, filiais, clientes, fornecedores, parceiros comerciais e usuários. É caracterizado pela integração dos mais diversos sistemas de diferentes organizações, nos quais as partes envolvidas cooperam entre si, de modo a buscar um objetivo em comum. Ainda segundo (DUMONT, 2006), as principais técnicas de segurança são: adoção de uma boa política de segurança, utilização de um firewall, implantação de sistemas de detecção e prevenção de intrusos, criptografia, autenticação, configuração correta do sistema, monitoramento e verificação de integridade do sistema, definição de políticas de contingência, *backup*, além de diversos outros elementos, possibilitando a proteção do sistema em diversos níveis.

2.5 Mecanismos de Segurança com Lógica Nebulosa

Em *A Novel Approach to Manage Trust in Ad Hoc Networks* (ZHOU, 2007), é apresentado um modelo de confiança baseado em inferência nebulosa para melhorar a

segurança de redes ad hoc. Foi utilizada uma característica importante, que é a incorporação da maioria dos conceitos essenciais para a confiança, como a subjetividade, imprecisão e incerteza. Simulações provaram que o modelo proposto não só atinge alta precisão e boa adaptabilidade na avaliação de confiança, mas também se apresenta robusto contra ataques.

2.6 Gestão de Projetos com Lógica Nebulosa

Em (WALKER, 2006) é apresentada uma técnica e uma ferramenta de *software* para a determinação e comunicação de risco técnico de mudanças em produtos de *software*. Os autores buscam capturar o risco de falhas geradas no processo de desenvolvimento de *software* usando como premissas as dependências funcionais dos módulos de *software*, seu histórico de atualizações, e o conhecimento dos analistas sobre os componentes que são afetados pelas mudanças.

Em (TAH, 2000) é proposto o uso da lógica difusa na determinação de risco de projetos de construção. O estudo lista os riscos e cria funções de pertinência para a determinação da possibilidade de ocorrências (extremamente alta, muito alta, alta, média, baixa, muito baixa e extremamente baixa) dos referidos riscos, e de suas severidades (muito alta, alta, média, baixa e muito baixa). Regras são propostas relacionando riscos, possibilidades, severidades e suas consequências em termos de custo, qualidade, tempo e segurança do projeto.

2.7 Comentários

Este capítulo procurou expor os trabalhos pertinentes aos temas abordados nesta dissertação. Houve uma maior preocupação em enfatizar estudos relacionados à segurança, e em especial aos sistemas de detecção de intrusos.

No próximo capítulo, será feita a apresentação da modelagem utilizada no estudo de caso, isto é, um sistema distribuído que visa acompanhar o desenvolvimento de um *software* e seus componentes, o Gestor de *Software* na Nuvem e o Nodo de *Software* na Nuvem.

3 MODELAGEM

3.1 Introdução

Neste capítulo é abordada toda a modelagem do estudo de caso desenvolvido, ou seja, um sistema distribuído para acompanhar o desenvolvimento de um produto de *software*. Tal sistema possui um programa central, chamado de Gestor de *Software* na Nuvem; e outro programa que é executado pelos analistas colaboradores chamado de Nodo de *Software* na Nuvem.

Consultando um especialista, pode ser constatado que nem sempre o cenário que apresenta uma máquina sem acesso é resultado de um problema de rede.

A escolha da utilização da função da variável linguística triangular deve-se ao fato de ser mais razoável para execução do processo de defuzzificação. A variação pequena no cálculo, e ainda, seu custo reduzido, de modo a simplificar e apresentar-se menos complexo no que se refere à implementação foram fatores decisivos para seguir o triângulo para calcular o segmento de reta, em detrimento às funções não lineares (como a gaussiana). A função triangular possui um ponto máximo, e, de acordo com o especialista consultado, para cada termo há um único ponto no universo de discurso com pertinência máxima, ficando claro que este deveria ser o modelo adotado.

Na próxima seção, apresenta-se a arquitetura de todo o sistema, seguindo com a descrição da análise do Gestor de *Software* na Nuvem na terceira seção. Depois, na quarta seção, é apresentada a análise do Nodo de *Software* na Nuvem, enquanto que a quinta e a sexta seções mostram os comentários sobre os protocolos utilizados pelo Gestor e pelo Nodo de *Software* na Nuvem, respectivamente. Por último, a sétima seção exhibe a confecção do modelo dos sistemas de inferência nebulosos para a rede e para a segurança, os quais são executados pelo Gestor de *Software* na Nuvem.

3.2 Arquitetura do Estudo de Caso

No sentido de validar a abordagem de segurança para nuvem com lógica nebulosa proposta nesta dissertação, foi implementado um estudo de caso que tem como função gerir o desenvolvimento de um produto de *software* de forma distribuída. O ambiente considerado trata-se de várias pessoas (analistas e/ou programadores) que interagem através da nuvem

para construir um produto de *software* de forma colaborativa. Todos esses colaboradores são coordenados por um gerente que monitora e avalia o trabalho de desenvolvimento do *software* pela nuvem também.

Uma vez que o estudo de caso em questão consistiu em uma aplicação distribuída para o desenvolvimento de *software*, cada programa Nodo (usado pelo colaborador) na nuvem seria o responsável pela criação dos artefatos de uma fase do projeto do *software*, como, por exemplo, a fase de análise que poderia ser concebida por um colaborador e, sempre que os artefatos dessa análise estivessem concluídos (diagramas e especificações), eles seriam enviados ao programa Gestor (usado pelo gerente). Observe a Figura 10, a seguir, que evidencia a arquitetura da aplicação do estudo de caso.

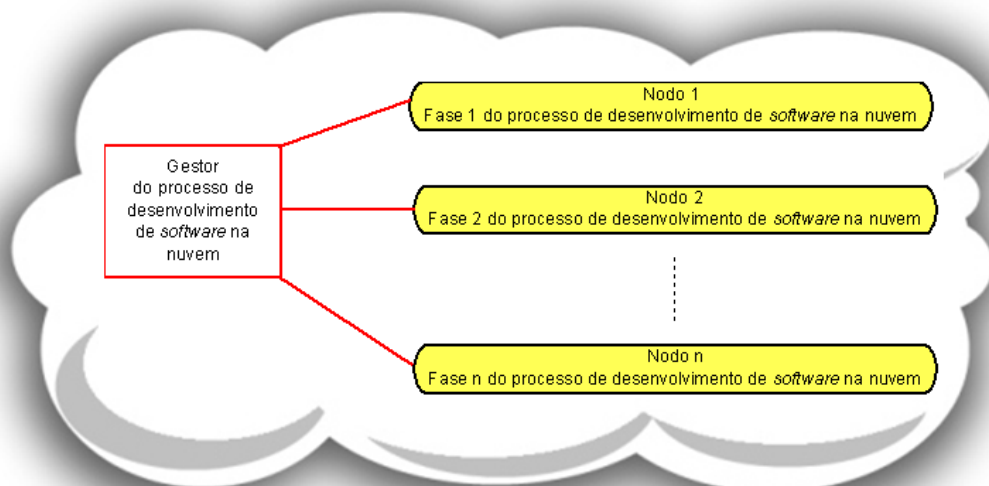


Figura 11 – Arquitetura do Sistema Distribuído para Desenvolvimento de *Software* na Nuvem.

Na abordagem proposta, a segurança na nuvem é obtida por meio da execução de dois sistemas de inferência nebulosos que são executados pelo programa Gestor do sistema distribuído que compõe o estudo de caso. Esses sistemas nebulosos são empregados para avaliar a situação da segurança na nuvem, uma vez que a aplicação distribuída só será segura se todos os seus nós estiverem seguros. Se apenas um de seus nós tiver sido atacado, significa que o trabalho cooperativo desempenhado pela aplicação na nuvem não será concluído.

O programa Gestor de *Software* na Nuvem será o responsável pelo monitoramento de todos os nós do estudo de caso. Ele se comunicará com cada programa Nodo de *Software* na Nuvem para inferir se esse possui ou não um problema de segurança. Uma vez que a comunicação entre o Gestor e o Nodo pode simplesmente falhar por problemas na rede sem

que haja problema de segurança, foi concebido, portanto, dois sistemas de inferência nebulosos: um para inferir problemas na rede, chamado de Sistema de Inferência de Rede (SIR); e um outro para, de fato, deduzir se ocorreu um ataque ao nó, chamado de Sistema de Inferência de Segurança (SIS).

Já que o estudo de caso é composto por dois programas independentes (o Gestor e o Nodo) que interagem através da nuvem, nas próximas seções serão apresentadas as modelagens de cada um deles em separado.

3.3 Análise do Gestor de *Software* na Nuvem

A análise aqui apresentada do programa Gestor de *Software* na Nuvem (gestsoftnuvem) é composta do modelo de casos de uso (subseção 3.3.1) que retrata as suas funcionalidades e do modelo de classes que expõe a sua estrutura interna (subseção 3.3.2). Tais modelos são exibidos por meio dos diagramas da UML.

Para a confecção dos diagramas UML do programa gestsoftnuvem foi utilizada a ferramenta BoUML. O programa gestsoftnuvem foi desenvolvido de acordo com a prática de programação em camadas, baseada na arquitetura *Model, View, Controler* (MVC), a qual favorece a manutenção de código e permite um projeto melhor estruturado, pois há menos entrelaçamento de código entre as classes do programa. A Figura 12 exibe o diagrama de execução do Gestor de *Software* na Nuvem, na qual pode-se observar as várias camadas do programa representadas, cada uma, por um fragmento.

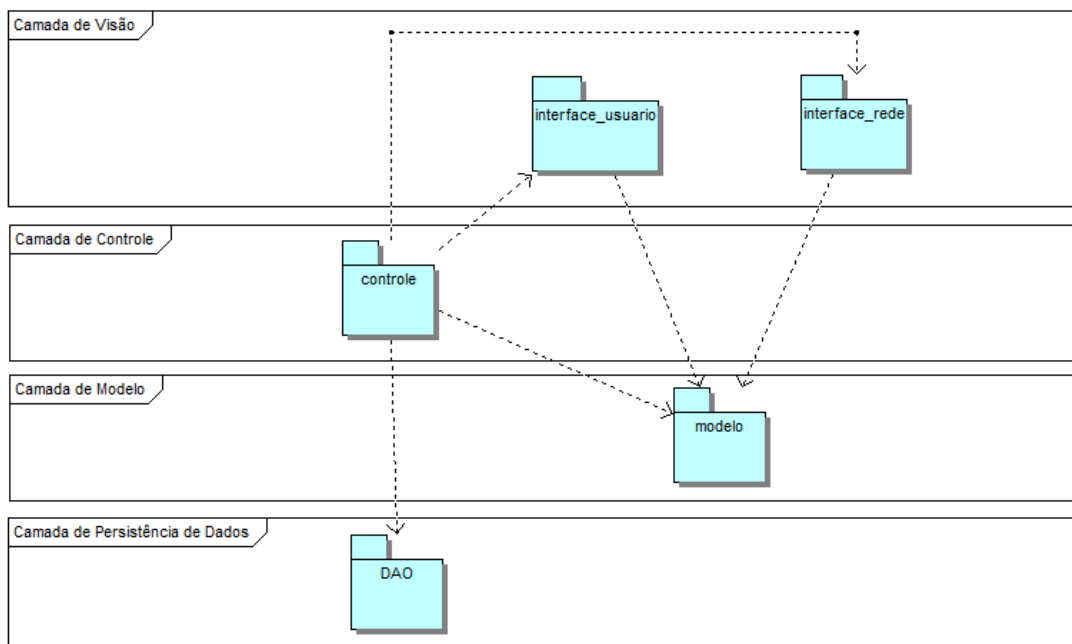


Figura 12 – Arquitetura do Gestor de *Software* na Nuvem.

Os fragmentos do `gestsoftnuvem` exibidos pela Figura 12, caracterizadores de cada camada da sua implementação, contêm todos os seus pacotes, a saber:

- Pacote **interface_usuario** → contém a implementação das classes responsáveis por interagir com o usuário do `gestsoftnuvem` através de janelas e eventos percebidos nos objetos de tela;
- Pacote **interface_rede** → contém a implementação das classes responsáveis pela comunicação entre o programa Gestor e o programa Nodo de *Software* na Nuvem;
- Pacote **controle** → contém a implementação das classes responsáveis pela interação entre as várias camadas, tendo a tarefa de disparar toda a funcionalidade do `gestsoftnuvem`;
- Pacote **modelo** → contém a implementação das classes cujos objetos são responsáveis por manter em memória os dados de cada registro de cada entidade do banco de dados utilizado pelo `gestsoftnuvem`, tratando-se apenas de uma espécie de memória auxiliar para receber os dados que vêm do banco ou para prepará-los na ocasião de serem persistidos no banco;
- Pacote **Data Access Object (DAO)** → contém a implementação das classes responsáveis pela execução das operações de inserção, consulta, atualização e exclusão de registros nas tabelas do banco de dados do `gestsoftnuvem`.

A seguir, é apresentado o modelo de casos de uso que expõe as funcionalidades do *gestsoftnuvem*.

3.3.1 Modelo de Casos de Uso

O programa Gestor de *Software* na Nuvem possui basicamente duas grandes tarefas, a saber: (1) oferecer ao usuário a possibilidade de manipular as informações do seu banco de dados; e, (2) executar o monitoramento das máquinas dos colaboradores para averiguar se os mesmos apresentam problemas de segurança. A Figura 13 apresenta o diagrama de casos de uso, feito no BoUML, com todas as funcionalidades do *gestsoftnuvem*. Nesta figura os seguintes casos de uso são observados:

- **manter_equipe**→ com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar, inserir, atualizar e excluir registros da tabela Equipe do banco de dados. Tal tabela mantém o cadastro dos analistas e programadores que integram o corpo de desenvolvedores dos produtos de *software* em construção, os quais estão sob controle do Gestor;
- **manter_sistema**→ com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar, inserir, atualizar e excluir registros da tabela Sistema do banco de dados. Tal tabela mantém o cadastro dos produtos de *software* em construção, os quais estão sob controle do Gestor;

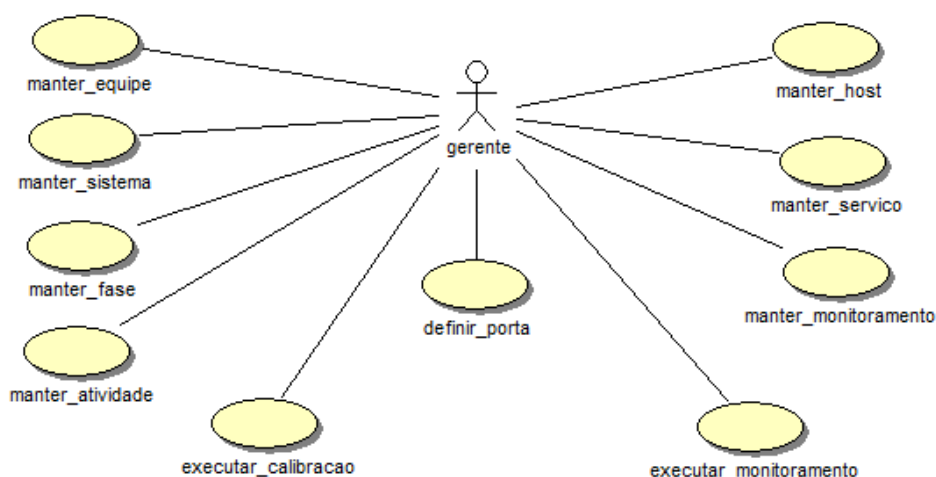


Figura 13 - Modelo de Caso de Uso para o Gestor de *Software*

- **manter_fase** → com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar, inserir, atualizar e excluir registros da tabela Fase do banco de dados. Tal tabela mantém o cadastro das fases (etapas ou partes) dos produtos de *software* em construção, as quais se encontram em desenvolvimento por algum colaborador na nuvem;
- **manter_atividade** → com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar e atualizar os registros da tabela Atividade do banco de dados. Tal tabela mantém o cadastro das informações dos arquivos dos artefatos enviados ao programa Gestor pelos desenvolvedores dos produtos de *software* em construção. Essas informações são compostas do endereço IP do *host* que enviou o artefato para o Gestor, do *status* de tal artefato (ver Seção 4.2) e do nome completo, incluindo o caminho do arquivo do artefato. Esse arquivo trata-se do resultado da atividade realizada pelo colaborador ao executar a fase que lhe coube da implementação do *software* em desenvolvimento. A inserção de registros na tabela Atividade não é permitida ao usuário do Gestor, pois ela é feita de forma automática. A exclusão também não é permitida para não se perder o histórico de tudo que foi feito pelos colaboradores;
- **manter_host** → com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar, inserir, atualizar e excluir registros da tabela Host do banco de dados. Tal tabela mantém o cadastro das máquinas dos colaboradores dos produtos de *software* em construção, os quais estão sob controle do Gestor;
- **manter_servico** → com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar, inserir, atualizar e excluir registros da tabela Servico (ver Seção 4.2) do

banco de dados. Tal tabela mantém o cadastro dos serviços presentes nas máquinas dos colaboradores dos produtos de *software* em construção, os quais estão sob controle do Gestor;

- **manter_monitoramento**→ com essa funcionalidade, o usuário do *gestsoftnuvem* pode consultar e excluir registros da tabela Monitoramento (ver Seção 4.2) do banco de dados. Tal tabela mantém os registros do resultado do monitoramento que o Gestor faz nas máquinas dos colaboradores dos produtos de *software* em construção. É exatamente nessa tabela que ficam armazenados os valores inferidos pelos sistemas nebulosos (de rede e de segurança) processados após o *gestsoftnuvem* obter os dados das entradas para esses sistemas relativos a cada máquina dos nodos. Em tal tabela, só é permitida a consulta e exclusão, pois a inserção é automática e a atualização pelo usuário não faria sentido, já que os valores dos campos são parâmetros de rede medidos com a nuvem em funcionamento;
- **executar_monitoramento**→ com essa funcionalidade, o usuário do *gestsoftnuvem* pode dar início ao processo de monitoramento dos nodos cadastrados para averiguar se os mesmos estão funcionando normalmente e se sofreram ou não ataques. O usuário do programa Gestor pode interromper o processo de monitoramento a qualquer momento;
- **executar_calibracao**→ com essa funcionalidade, o usuário do *gestsoftnuvem* pode dar início ao processo de calibração dos nodos cadastrados para obter valores iniciais de alguns parâmetros de rede para, posteriormente, durante o monitoramento, comparar esses valores com outros novos medidos de tal forma a inferir se houve ou não um problema de rede ou de segurança com os nodos. Tais valores obtidos na calibração ficam armazenados na tabela Host do banco de dados do *gestsoftnuvem*;
- **definir_porta**→ com essa funcionalidade, o usuário do *gestsoftnuvem* simplesmente escolhe o número da porta de Internet com a qual a classe *Servidor* do programa irá funcionar.

3.3.2 Modelo de Classes

A Figura 14 apresenta o modelo de classes do *gestsoftnuvem*. Esta figura mostra as classes organizadas dentro de seus pacotes.

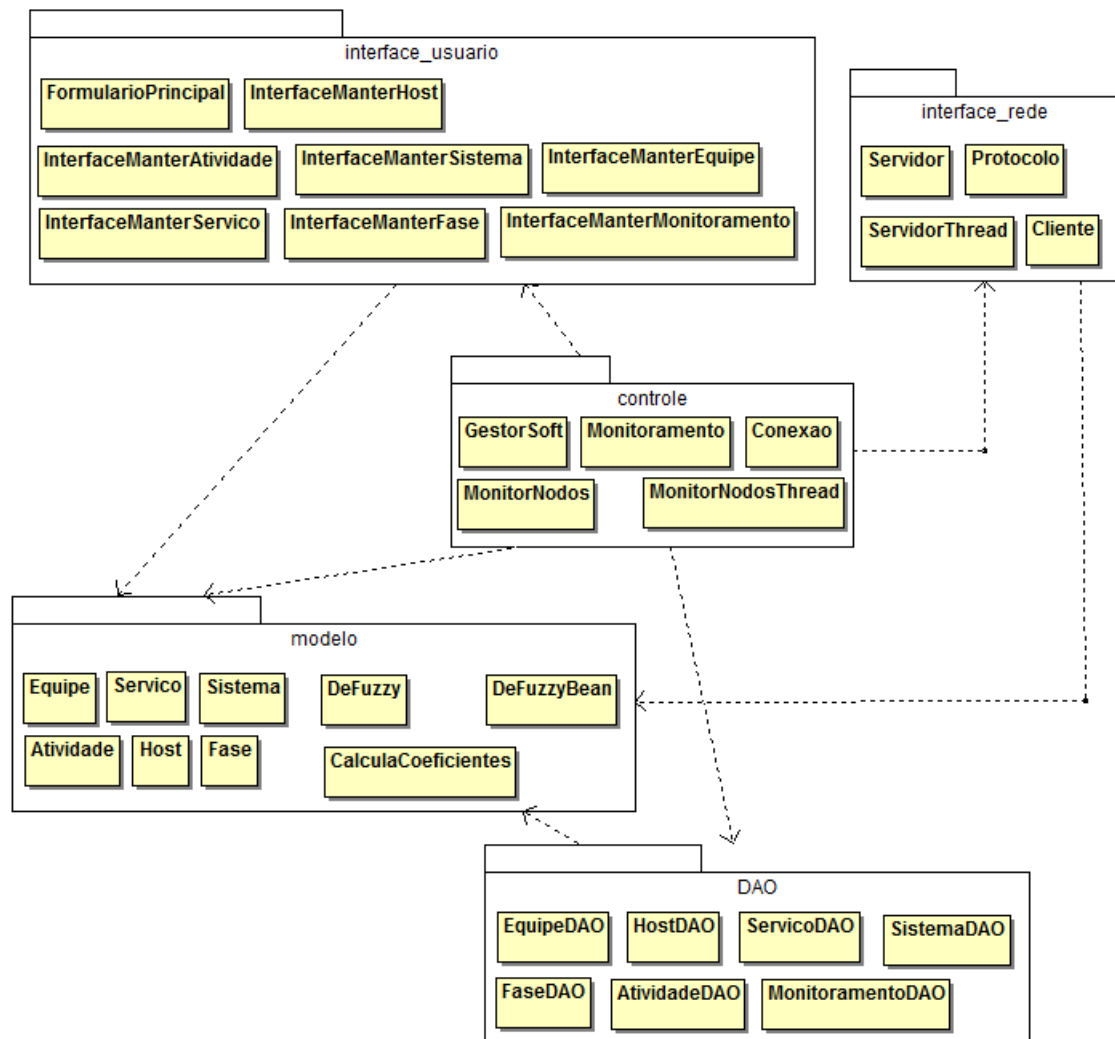


Figura 14- Modelo de Classes do Gestor de *Software* na Nuvem

No pacote **interface_usuario** da Figura 14, encontra-se a classe principal do programa (*FormularioPrincipal*) e mais uma classe específica para tratar a interface de tela de cada entidade do banco de dados. O pacote **interface_rede** contém as classes responsáveis pelo processamento da comunicação entre o programa Gestor e o programa Nodo utilizado pelos colaboradores do *software* em desenvolvimento.

O pacote **modelo** contém as classes para manipulação dos registros das entidades do banco de dados, além das classes *DeFuzzy*, *DeFuzzyBean* e *CalculaCoeficientes* que processam a inferência dos sistemas nebulosos de rede e segurança. O pacote DAO possui uma classe específica para processar a manutenção de cada entidade do banco de dados.

Por fim, o pacote **controle** contém as classes que integram todo o programa, promovendo o acionamento das funcionalidades do Gestor. Uma explanação mais detalhada

de cada uma dessas classes e de todos os seus métodos integrantes pode ser vista no Capítulo 4. A seguir, será apresentada a análise do programa *Nodo de Software* na Nuvem.

3.4 Análise do Nodo de *Software* na Nuvem

Assim como na análise do *gestsoftnuvem*, a modelagem aqui apresentada do programa *Nodo de Software* na Nuvem (*nodotsoftnuvem*) é composta do modelo de casos de uso (Subseção 3.4.1) que retrata as suas funcionalidades e do modelo de classes que expõe a sua estrutura interna (Subseção 3.4.2). O programa *nodotsoftnuvem* também foi desenvolvido de acordo com a prática de programação em camadas, seguindo a arquitetura MVC. A Figura 15 exhibe a arquitetura do *Nodo de Software* na Nuvem.

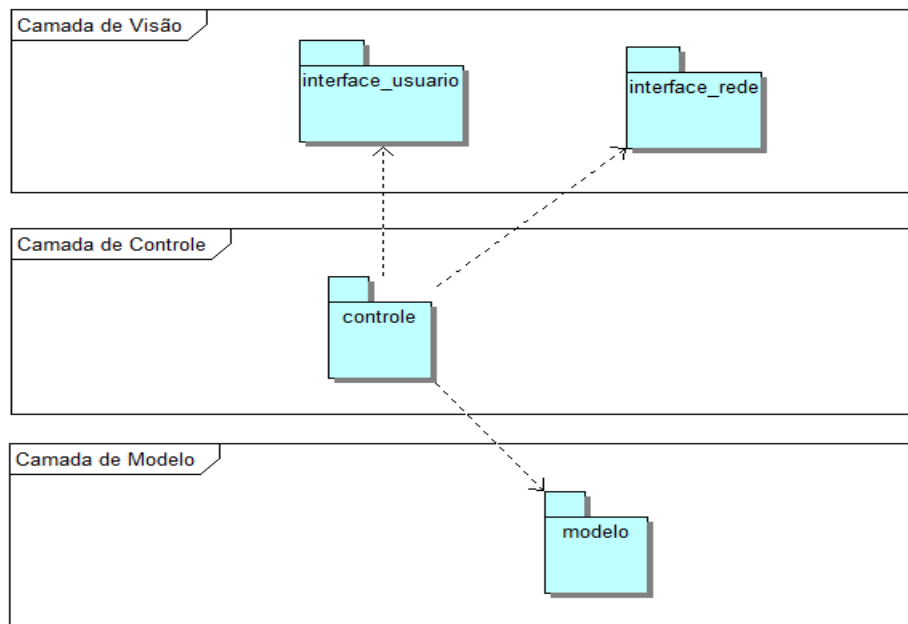


Figura 15 – Arquitetura do *Nodo de Software* na Nuvem.

As camadas do *nodotsoftnuvem*, exibidas pela Figura 15, contêm todos os seus pacotes, a saber:

- Pacote **interface_usuario** → possui a implementação da classe responsável por interagir com o usuário do *Nodo* através de janela e eventos percebidos nos objetos de tela;
- Pacote **interface_rede** → contém a implementação das classes responsáveis pela comunicação com o programa *gestsoftnuvem*;

- Pacote **controle** → contém a implementação da classe responsável pela interação entre as camadas de Visão e de Modelo, tendo a tarefa de comandar a execução do *nodosoftnuvem*;
- Pacote **modelo** → contém apenas uma classe chamada *Gestor* para manter em memória, durante a execução do programa, as informações sobre o número de porta e o endereço IP do programa *gestsoftnuvem*.

A seguir, é apresentado o modelo de casos de uso que expõe as funcionalidades do *nodosoftnuvem*.

3.4.1 Modelo de Casos de Uso

O programa *Nodo de Software na Nuvem* foi concebido para ser utilizado pelo colaborador e interagir com o *gestsoftnuvem*. A Figura 16 apresenta o diagrama de casos de uso, com a funcionalidade do *nodosoftnuvem*. Nesta figura, os seguintes casos de uso são observados:

- **definir_porta_nodo** → com essa funcionalidade, o usuário do *nodosoftnuvem* especifica o número da porta de Internet com a qual a classe *Servidor* do programa irá funcionar;
- **definir_porta_gestor** → com essa funcionalidade, o usuário do *nodosoftnuvem* especifica o número da porta de Internet a qual o programa irá utilizar para se comunicar com o *gestsoftnuvem*;
- **definir_ip_gestor** → com essa funcionalidade, o usuário do *nodosoftnuvem* especifica o endereço IP do computador do Gestor de *Software* na Nuvem;
- **enviar_artefato** → com essa funcionalidade, o usuário do *nodosoftnuvem* escolhe um arquivo (do tipo PDF) que é enviado para o *gestsoftnuvem*. Tal arquivo trata-se de um artefato (diagrama, arquivo fonte, documentação etc.) elaborado pelo colaborador do *software* em desenvolvimento.

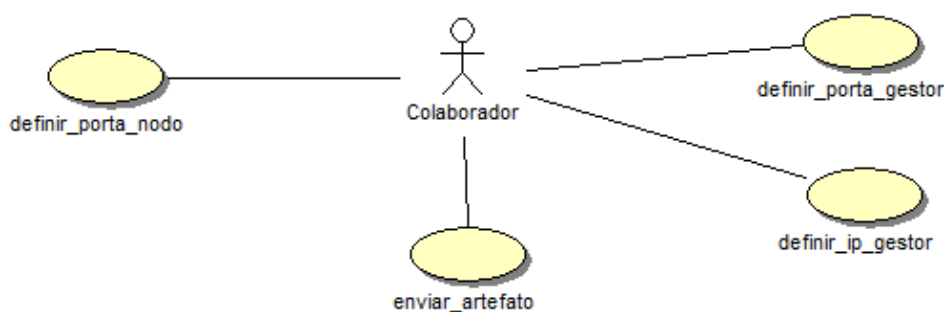


Figura 16 - Modelo de Casos de Uso do Nodo de *Software* na Nuvem

3.4.2 Modelo de Classes

A Figura 17 apresenta o modelo de classes do *nodosoftnuvem*. Esta figura mostra as classes com a sua especificação completa (nome, atributos e métodos), diferentemente do diagrama de classes do *gestsoftnuvem*, que apresentou as classes com a sua especificação simplificada (apenas nome) para não prejudicar a sua visualização.

A Figura 17 mostra a classe principal do programa Nodo (*InterfacePrincipal*) que é a única do pacote **interface_usuario**. Essa classe é responsável pela exibição da tela do *nodosoftnuvem* e recebe os eventos de tela provocados pelo seu usuário, procedendo com a execução dos métodos que atendem a cada evento. Esses métodos são implementados na classe *ControlaNodo*, sendo a única do pacote **controle**; e é a responsável pela integração de todas as camadas do *nodosoftnuvem*.

As classes *Servidor*, *Cliente* e *Protocolo* compõem o pacote **interface_rede**, sendo responsáveis pela comunicação com o programa *gestsoftnuvem*. A classe *Cliente* possui o código encarregado de tomar a iniciativa de comunicação com o *gestsoftnuvem* para enviar o arquivo do artefato. Já a classe *Servidor* fica aguardando uma solicitação do *gestsoftnuvem* quando ele envia uma mensagem ao Nodo para calibrá-lo ou monitorá-lo.

A classe *Protocolo* possui a implementação da regra de comunicação obedecida pelo Nodo e pelo Gestor no momento em que eles precisam se comunicar. A especificação do protocolo de comunicação empregado por esses programas é explanada nas seções a seguir e no Capítulo 4, existem mais detalhes sobre os métodos e classes do *nodosoftnuvem*.

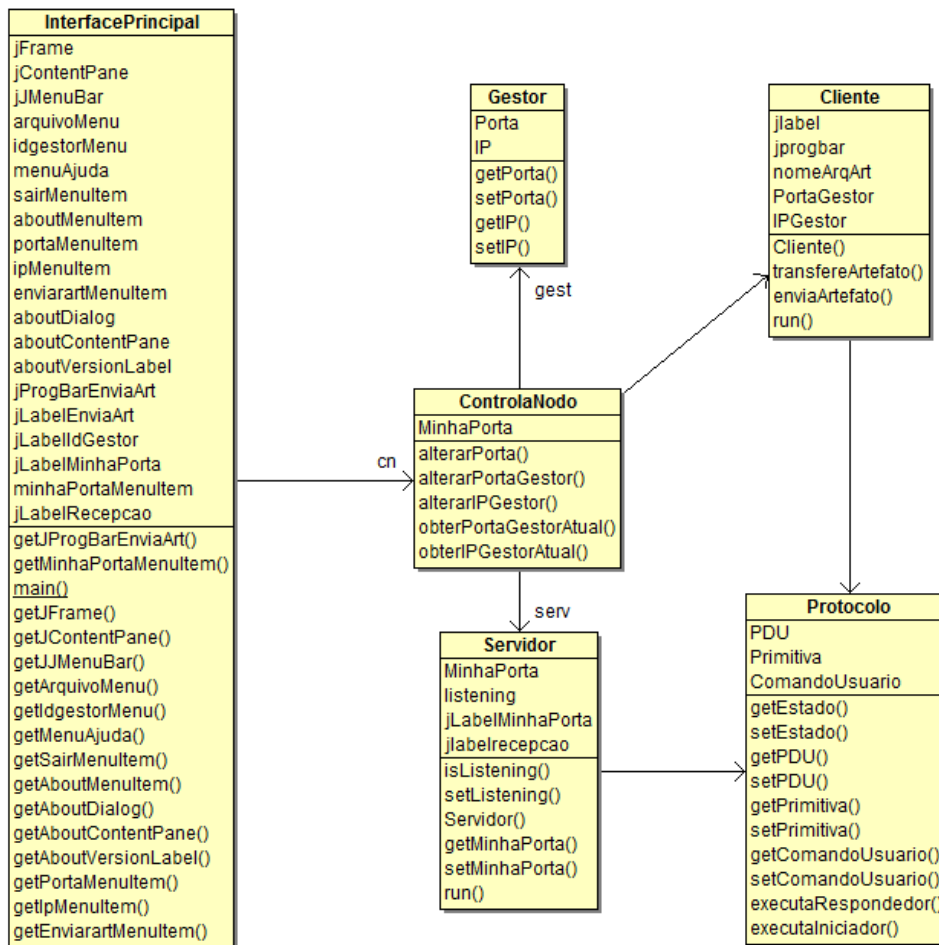


Figura 17 - Modelo de Classes do Nodo de *Software* na Nuvem

3.5 Protocolo do Gestor de *Software* na Nuvem

A classe *Protocolo* do pacote **interface_rede** do `gestsoftnuvem` possui a implementação do protocolo de comunicação usado pelo Gestor para se comunicar com o Nodo. Tal protocolo é usado pelo Gestor, no papel de iniciador da comunicação, para calibrar e monitorar o Nodo. Já no papel de respondedor da comunicação, o protocolo empregado possibilita ao Gestor receber o arquivo de artefato enviado pelo Nodo.

As mensagens usadas pelo Gestor (no papel de iniciador) para calibrar o Nodo são chamadas de PING e PONG. O Gestor envia uma *Protocol Data Unit* (PDU) PING para o Nodo, e este responde com a PDU (mensagem) PONG. Ocorre um atraso entre o envio de um PING e a recepção de um PONG, o qual é medido e armazenado em milissegundos na tabela Host, no registro correspondente ao Nodo calibrado.

O Gestor também executa uma varredura de portas dos serviços cadastrados como ativos na tabela *Servico*, completando assim, o processo de calibração do Nodo. Na Figura 18 é apresentado o diagrama de estados da UML com o funcionamento do protocolo executado pelo Gestor.

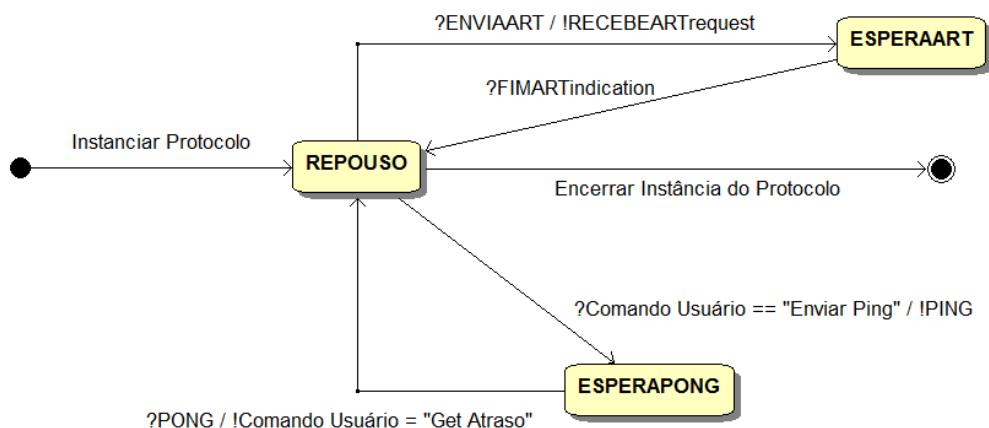


Figura 18 – Máquina de Estados do Protocolo do Gestor de *Software* na Nuvem.

Na Figura 18, observam-se os símbolos de pontuação ? e ! que indicam, respectivamente, recepção e transmissão que podem ser de mensagens (PDU), comando do usuário do protocolo (método *getAtraso* da classe *Cliente*) ou primitiva de serviço. As primitivas de serviço são comandos trocados entre a classe *Protocolo* e a classe *Cliente*.

A Figura 18 mostra os dois comportamentos do Gestor, ou seja, o papel iniciador e o papel respondedor. Nesse último papel, o Gestor sai do seu estado REPOUSO quando recebe a PDU ENVIART que foi transmitida por um Nodo. O Gestor faz a transição para o estado ESPERAART, enviando a primitiva RECEBEARTrequest para a classe *Cliente* que processa a recepção byte a byte do arquivo de artefato transmitido por um Nodo. Com a conclusão dessa recepção, a classe *Cliente* passa para a classe *Protocolo* a primitiva FIMARTindication que provoca a transição do Gestor de volta para o seu estado de REPOUSO.

3.6 Protocolo do Nodo de *Software* na Nuvem

A classe *Protocolo* do pacote *interface_rede* do *nodosoftnuvem* possui a implementação do protocolo de comunicação usado pelo Nodo para se comunicar com o Gestor. Tal protocolo é usado pelo Nodo, no papel de iniciador da comunicação, para enviar

um arquivo de artefato ao Gestor. Já no papel de respondedor da comunicação, o protocolo empregado possibilita ao Nodo receber as mensagens PING, respondendo com PONG, para que o Gestor obtenha a medida do atraso tanto na calibração quanto no monitoramento.

A Figura 19 apresenta o diagrama de estados da UML com o funcionamento do protocolo executado pelo Nodo na Modelagem dos Sistemas de Interferência Nebulosos.

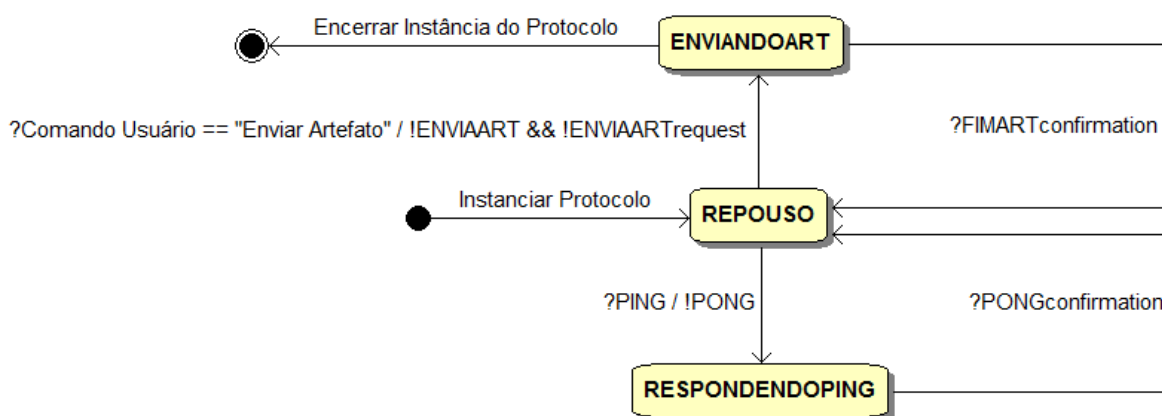


Figura 19 – Máquina de Estados do Protocolo do Nodo de *Software* na Nuvem.

A Figura 19 mostra que o protocolo do Nodo sai do seu estado de REPOUSO quando recebe uma PDU PING, que é prontamente respondida com a mensagem PONG. Nesse momento o protocolo transita para o estado RESPONDENDOPING que, por sua vez, é deixado pelo Nodo ao receber da sua classe *Servidor* a primitiva PONGconfirmation.

Outra situação do Nodo evidenciada pela Figura 19 é quando o protocolo recebe o comando “Enviar Artefato” de seu usuário (método *enviaArtefato* da classe *Cliente*). Nesse momento, o protocolo envia a PDU ENVIAART por meio do pedido da primitiva de serviço ENVIAARTrequest à classe *Cliente*. O protocolo então migra para o estado ENVIANDOART, e, deixa o mesmo após receber da classe *Cliente* a primitiva FIMARTconfirmation, retornando finalmente ao estado de REPOUSO.

3.7 Modelagem dos Sistemas de Interferência Nebulosos

A abordagem para a detecção de intrusos, proposta nesta dissertação, baseia-se na ideia de que uma aplicação executada na nuvem precisa garantir que todos os seus nós estejam em perfeito funcionamento, para que o trabalho cooperativo tenha efetividade. Sendo assim, tal abordagem prevê um monitoramento de todos os componentes envolvidos na aplicação em

nuvem de forma periódica, para se detectar previamente um problema em qualquer um desses componentes, antes que ele comprometa de forma crítica todo o trabalho.

O monitoramento de todos os nós da aplicação em nuvem para garantir a sua segurança é exatamente o que caracteriza a principal contribuição da abordagem proposta de detecção de intrusos, visto que a maioria dos mecanismos de segurança consagrados são implantados em uma única máquina *host*.

Levando-se em conta que o monitoramento dos nodos da nuvem envolve comunicação em rede, é necessário descartar qualquer problema de rede antes de se concluir que o nodo tenha sido atacado. Nesse caso, a abordagem de detecção de intrusos na nuvem foi composta por dois sistemas de inferência nebulosos, a saber:

- **Sistema de Inferência de Rede (SIR)** → responsável por avaliar a existência de problemas na comunicação com um nodo da nuvem, a fim de descartar conclusões equivocadas a respeito desse nodo possuir problemas de segurança;
- **Sistema de Inferência de Segurança (SIS)** → responsável por avaliar a existência de problemas de segurança em um nodo da nuvem. Essa inferência toma como entrada a saída do SIR, pois, no caso desse último indicar problemas de comunicação, não é possível deduzir positivamente se o nodo sofreu ou não um ataque.

A justificativa da criação desses dois sistemas de inferência nebulosos, indicados por um especialista em segurança de redes de computadores, reside no fato de alguns tipos de ataques, como o *deny of service* (DOS), deixar o *host* atacado sem acesso pela rede, todavia o *host* sob análise pode de fato estar sem acesso por possuir algum problema de rede (físico ou de configuração de software), sendo assim, deve-se procurar diferenciar quando um *host* está inacessível devido a um problema de segurança ou de rede.

A Figura 20 mostra esquematicamente as entradas e as saídas dos dois sistemas de inferência nebulosos os quais foram implementados no programa do estudo de caso gestsoftnuvem.

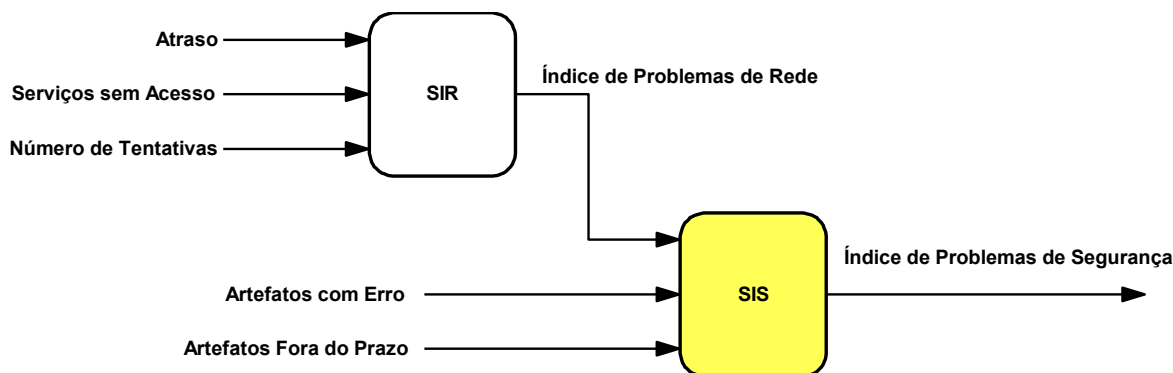


Figura 20 – Sistemas de Inferência Nebulosos para Detecção de Intrusos na Nuvem

As três variáveis linguísticas de entrada do SIR observadas na Figura 20 são: ATRASO, serviços sem acesso (SERVIÇOS) e número de tentativas (TENTATIVAS). O ATRASO é medido em milissegundos (ms), as TENTATIVAS em número de vezes e os SERVIÇOS em número de acessos. A Figura 20 ilustra também a saída, cuja resposta representa o Índice de Problemas de Rede (IPR) o qual indica, em porcentagem, a possível existência de problemas de rede em um nodo da nuvem.

A justificativa para o uso dessas três variáveis linguísticas de entrada para o SIS segue do raciocínio que vinha sendo construído pelo especialista desde a concepção das variáveis de entrada do SIR. Inicialmente, foi decidido que haveria um separação entre análise de problema de rede e análise de problema de segurança, então naturalmente o IPR seria uma das entradas do SIS, mas posteriormente, foi aventado pelo especialista que existem certos ataques como, por exemplo, Cavalos de Troia, que podem corromper os dados de um *host* localmente o que levaria à máquina a funcionar perfeitamente enviando dados, porém tais dados estariam adulterados. Para esse tipo de ataque, seria necessário atestar se o Gestor teria recebido dados corretos, então foi concebida a variável linguística Artefatos com Erro que mede a quantidade de dados recebidos com erro (adulterados) pelo Gestor.

Finalizando, para tipos de ataques, como o do *man in the middle*, (homem no meio) no qual os dados trocados entre duas partes, por exemplo, o Gestor e o Nodo, são de alguma forma interceptados, registrados e possivelmente alterados pelo atacante sem que as vítimas saibam o que está acontecendo. O atacante, nesse caso, pode decidir retransmitir os dados posteriormente o que acarretaria um atraso no prazo de entrega dessas informações. Para averiguar tal situação foi concebida a variável de entrada Artefatos Fora do Prazo para o SIS.

A Figura 21 apresenta as funções de pertinência da variável linguística ATRASO com termos nebulosos curto, médio e longo.

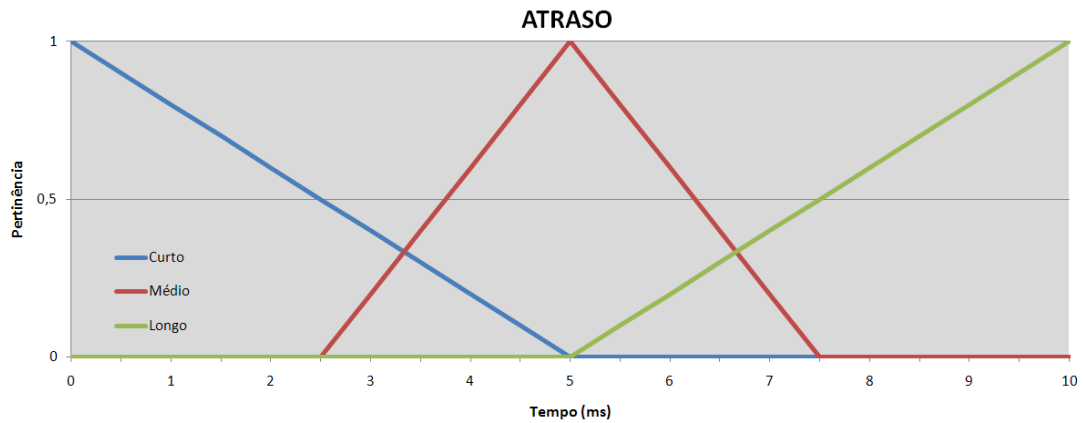


Figura 21 – Funções de Pertinência dos Termos da Variável Linguística ATRASO.

A Figura 22 apresenta as funções de pertinência da variável linguística TENTATIVAS com termos nebulosos pouco, médio e muito.

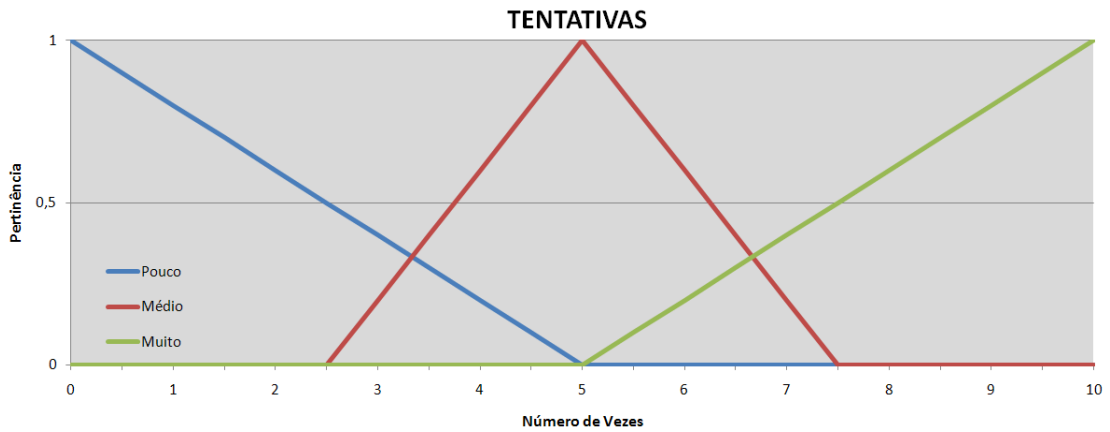


Figura 22 – Funções de Pertinência dos Termos da Variável Linguística TENTATIVAS.

A Figura 23 apresenta as funções de pertinência da variável linguística SERVIÇOS com termos nebulosos nenhum, alguns e todos.

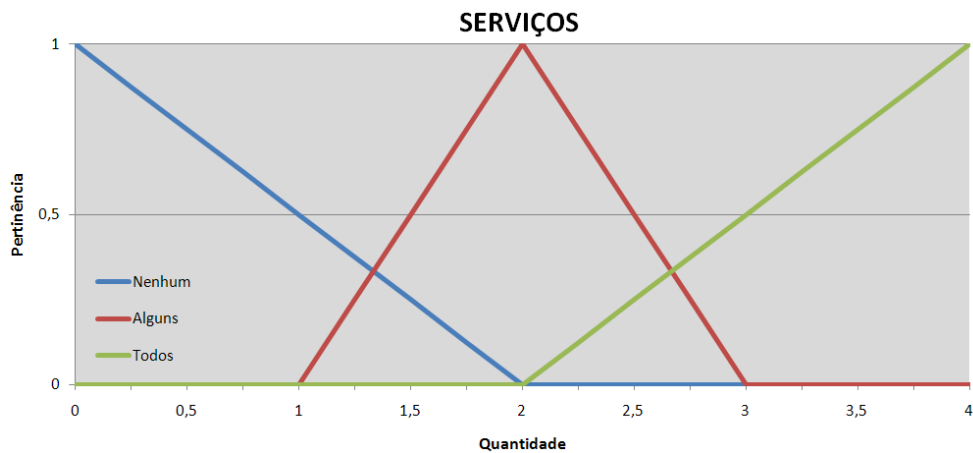


Figura 23 – Funções de Pertinência dos Termos da Variável Linguística SERVIÇOS.

A Figura 24 apresenta as funções de pertinência da variável linguística Índice de Problemas de Rede (IPR) com termos nebulosos sem problema, provável e com problema.

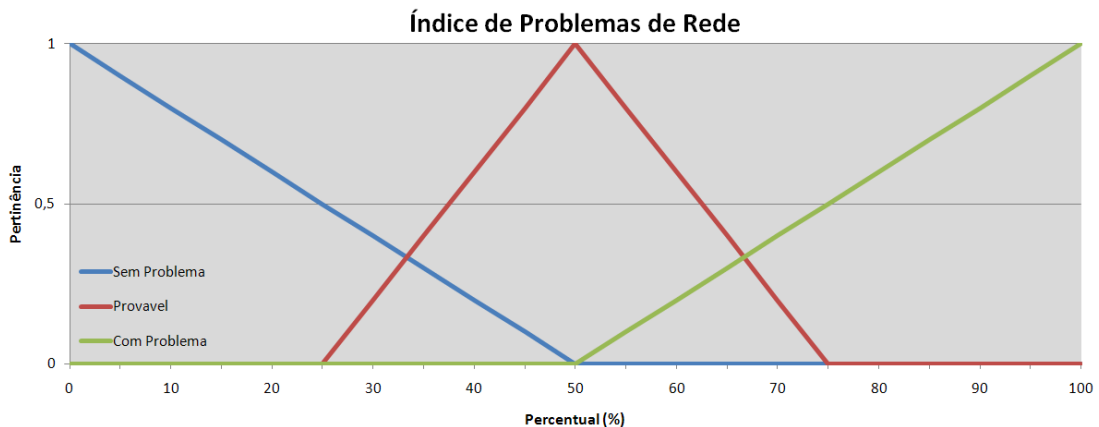


Figura 24– Funções de Pertinência dos Termos da Variável Linguística IPR.

A Figura 24 mostra também as três variáveis linguísticas de entrada do SIS, a saber: Índice de Problemas de Rede (IPR), Artefatos Fora do Prazo (AFP) e Artefatos com Erro (ACE). O SIS apresenta uma relação de dependência com o SIR, pois a sua entrada IPR é a saída do SIR. As outras duas entradas AFP e ACE, são medidas por percentual, em um universo de discurso de 0 a 100%. A saída do SIS é caracterizada pela variável linguística Índice de Problemas de Segurança (IPS).

A justificativa para o uso dessas três variáveis linguísticas de entrada para o SIS segue do raciocínio que vinha sendo construído pelo especialista desde a concepção das variáveis de entrada do SIR. Inicialmente, foi decidido que haveria um separação entre análise de problema de rede e análise de problema de segurança, então naturalmente o IPR seria uma das

entradas do SIS, mas posteriormente, foi aventado pelo especialista que existem certos ataques como, por exemplo, Cavalos de Troia, que podem corromper os dados de um *host* localmente o que levaria à máquina a funcionar perfeitamente enviando dados, porém tais dados estariam adulterados. Para esse tipo de ataque, seria necessário atestar se o Gestor teria recebido dados corretos, então foi concebida a variável linguística Artefatos com Erro que mede a quantidade de dados recebidos com erro (adulterados) pelo Gestor.

Finalizando, para tipos de ataques, como o do *man in the middle*, (homem no meio) no qual os dados trocados entre duas partes, por exemplo, o Gestor e o Nodo, são de alguma forma interceptados, registrados e possivelmente alterados pelo atacante sem que as vítimas saibam o que está acontecendo. O atacante, nesse caso, pode decidir retransmitir os dados posteriormente o que acarretaria um atraso no prazo de entrega dessas informações. Para averiguar tal situação foi concebida a variável de entrada Artefatos Fora do Prazo para o SIS.

As funções de pertinência da variável IPR (agora entrada para o SIS) com termos nebulosos sem problema, provável e com problema já foram apresentadas na Figura 24. A Figura 25 apresenta as funções de pertinência da variável linguística Artefatos Fora do Prazo (AFP) com termos nebulosos poucos, alguns e muitos.

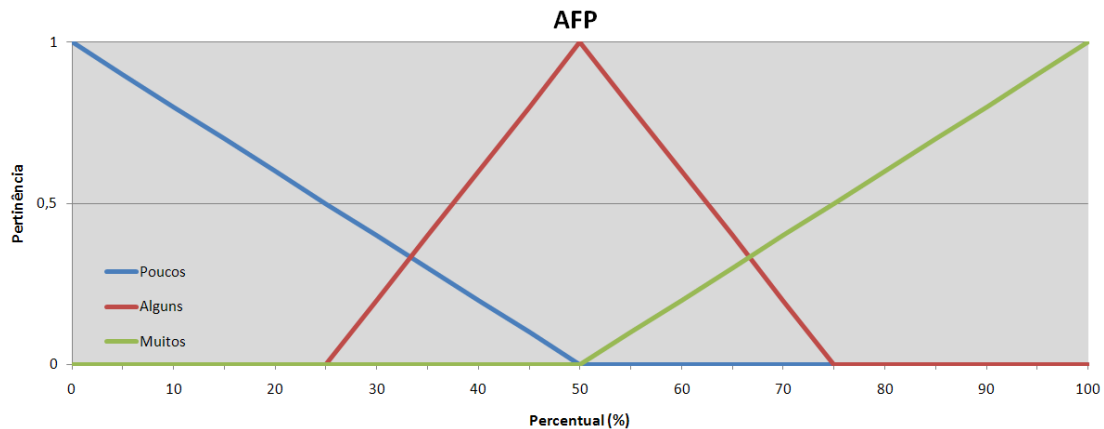


Figura 25 – Funções de Pertinência dos Termos da Variável Linguística AFP.

A Figura 26 apresenta as funções de pertinência da variável linguística Artefatos com Erro (ACE) com termos nebulosos poucos, alguns e muitos.

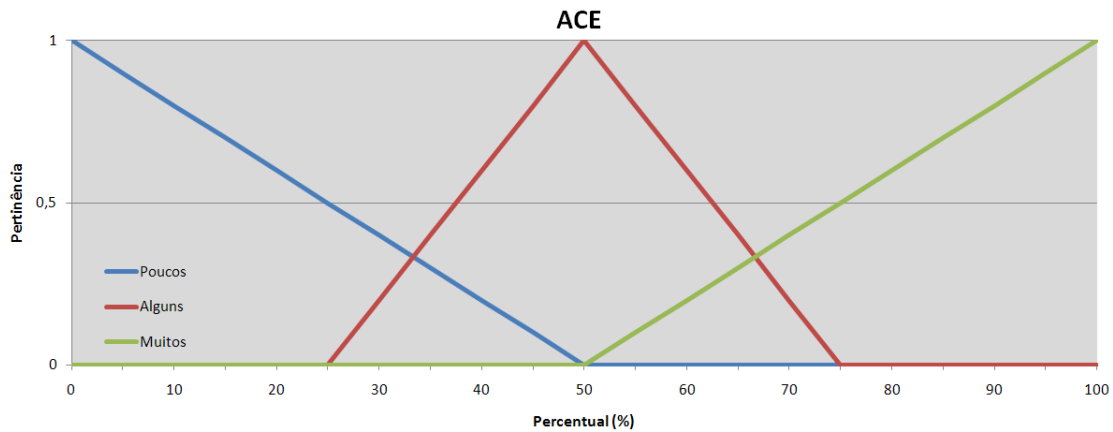


Figura 26 – Funções de Pertinência dos Termos da Variável Linguística ACE.

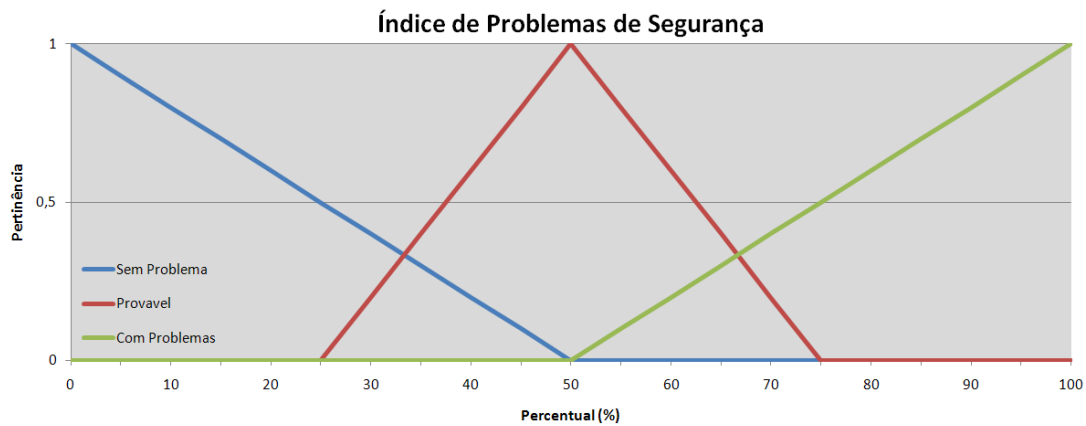


Figura 27 – Funções de Pertinência dos Termos da Variável Linguística IPS.

A Figura 27 apresenta as funções de pertinência da variável linguística Índice de Problemas de Segurança (IPS) com termos nebulosos sem problema, provável e com problema.

Todas as funções de pertinência de todos os termos de todas as variáveis linguísticas foram concebidas com a forma triangular, pois para o problema considerado tais termos possuem um único valor máximo de pertinência (de acordo com o especialista), ou seja, existe um único valor de atraso curto com pertinência 1,0, assim como existe um único valor com pertinência 1,0 para o atraso médio e assim por diante. Além disso, a função triangular, sendo composta por segmentos de reta, favorece o desempenho do processo de inferência assim como tornou a implementação mais simples e não ofereceu qualquer incoerência nos resultados.

As Tabelas 2 e 3 apresentam as regras de inferência dos dois sistemas nebulosos implementados (de rede e de segurança).

Tabela 2 – Regras de Inferência Nebulosas do SIR.

ATRASSO	TENTATIVAS	SERVIÇOS	IPR
CURTO	Pouco	Nenhum	Sem problema
		Alguns	Provável
		Todos	Provável
	Médio	Nenhum	Provável
		Alguns	Provável
		Muito	Provável
	Muito	Nenhum	Provável
		Alguns	Provável
		Todos	Com Problema
MÉDIO	Pouco	Nenhum	Provável
		Alguns	Provável
		Todos	Provável
	Médio	Nenhum	Provável
		Alguns	Provável
		Todos	Provável
	Muito	Nenhum	Provável
		Alguns	Provável
		Todos	Com Problema
LONGO	Pouco	Nenhum	Provável
		Alguns	Provável
		Todos	Provável
	Médio	Nenhum	Provável
		Alguns	Provável
		Todos	Provável
	Muito	Nenhum	Provável
		Alguns	Provável
		Todos	Com Problema

A Tabela 2 evidencia a influência homogênea das três variáveis linguísticas do SIR (ATRASSO, TENTATIVAS E SERVIÇOS) para se concluir que existe algum problema de comunicação com o nodo da nuvem, visto que basta o valor nebuloso de uma dessas variáveis ficar alto, para se inferir IPR *Com Problema*.

Tabela 3 – Regras de Inferência Nebulosas do SIS.

IPR	AFP	ACE	IPS
Sem Problema	Poucos	Poucos	Sem Problema
		Alguns	Provável
		Muitos	Provável
	Alguns	Muitos	Provável
		Alguns	Provável
		Muitos	Provável
	Muitos	Poucos	Provável
		Alguns	Provável
		Muitos	Com Problema
Provável	Poucos	Poucos	Provável
		Alguns	Provável
		Muitos	Provável
	Alguns	Poucos	Provável
		Alguns	Provável
		Muitos	Provável
	Muitos	Poucos	Provável
		Alguns	Provável
		Muitos	Com Problema
Com Problema	Poucos	Poucos	Provável
		Alguns	Provável
		Muitos	Provável
	Alguns	Poucos	Provável
		Alguns	Provável
		Muitos	Com Problema
	Muitos	Poucos	Provável
		Alguns	Com Problema
		Muitos	Com Problema

A Tabela 3 mostra o raciocínio para se concluir a existência de problemas de segurança, notando-se que no caso de ausência de problemas na rede, as variáveis linguísticas ACE e AFP passam a ser as determinantes do valor final do IPS. Por outro lado, quando se tem problemas de rede detectados, existem mais dúvidas sobre a segurança do nodo e, logo, não é possível inferir o valor *Sem Problema* para o IPS.

3.8 Comentários

A modelagem descrita neste capítulo foi projetada com o objetivo de acompanhar o desenvolvimento de um *software* em um sistema distribuído. Foram descritos detalhes da arquitetura do sistema e ainda os protocolos desenvolvidos para o Gestor de *Software* na Nuvem e para o Nodo de *Software* na Nuvem. Os sistemas de inferência desenvolvidos para os módulos de rede e segurança foram apresentados nas últimas seções deste capítulo.

Após terem sido expostos todos os conceitos do modelo proposto, no próximo capítulo serão comentados os aspectos da implementação do estudo de caso.

4 IMPLEMENTAÇÃO

4.1 Introdução

Neste capítulo serão abordados aspectos da implementação do estudo de caso desenvolvido. Na próxima seção é exposto o projeto físico do banco de dados por meio do esquema relacional das bases de dados que foram criadas. Na terceira Seção, é apresentada uma descrição breve de todos os métodos do programa Gestor de *Software* na Nuvem. Na quarta Seção, é explicado o algoritmo do modelo de inferência nebuloso adotado pelo SIR e pelo SIS. A quinta seção apresenta o algoritmo principal do programa Gestor de *Software* na Nuvem, o qual é responsável pelo monitoramento dos Nodos. A sexta Seção exibe com comentários as telas da interface homem-máquina do Gestor. A sétima Seção mostra a descrição dos métodos do programa Nodo de *Software* na Nuvem. Finalmente, a oitava Seção exibe as telas da interface homem-máquina do programa Nodo de *Software* na Nuvem.

4.2 Projeto Físico do Banco de Dados

O estudo de caso implementado utiliza duas bases de dados (gestsoftnuvem e lnebulosa) que foram criadas dentro do sistema gerenciador de banco de dados MySQL (MYSQL, 2013). A base de dados gestsoftnuvem é responsável por persistir as informações necessárias para controlar o desenvolvimento de um produto de *software* na nuvem, enquanto a base de dados lnebulosa possui os dados que definem os sistemas de inferência nebulosos utilizados pelo estudo de caso.

O esquema relacional destas bases de dados é apresentado a seguir:

Base de dados *gestsoftnuvem*:

atividade = (codigo: INT, host: TINYTEXT, status: TEXT, nome_arquivo_artefato: TEXT);

O campo *status* pode ser categorizado em:

- *Validando*: indica que o artefato foi recebido e está aguardando a validação por parte do usuário que deverá informar se esse artefato está correto ou errado, alterando para o caso em questão o valor do status;

- *Analisado*: indica que o artefato já foi considerado para uma etapa de monitoramento anteriormente já processada;
- *Atrasado*: indica que o artefato foi recebido com atraso de acordo com a data final da fase a qual ele se refere;
- *Correto*: indica que o artefato foi validado pelo usuário e considerado correto;
- *Errado*: indica que o artefato foi validado pelo usuário e considerado errado;
- *Atrasado e Errado*: indica que além do artefato ter sido recebido com atraso, foi validado pelo usuário o qual o considerou também com erro.

equipe = (codigo_funcionario: INT, nome_funcionario: TINYTEXT, cargo_funcionario: TINYTEXT, codigo_fase: INT);

fase = (nome_fase: TEXT, codigo_fase: INT, host: TEXT, status: TINYTEXT, data_inicio: DATE, data_final: DATE, codigo_sistema: INT);

host = (endereco_ip: CHAR, servicos: INT, atraso: DOUBLE);

monitoramento = (codigo: BIGINT, host: TEXT, atraso: DOUBLE, servicos: INT, tentativas: INT, indice_prob_rede: DOUBLE, artefatos_errados: INT, artefatos_atrasados: INT, indice_prob_seg: DOUBLE);

serviço = (codigo: INT, status: TEXT, host: TEXT, porta: INT);

O campo *status* pode ser categorizado em:

- *Fase*: indica que a porta desse serviço é a empregada para o Gestor se comunicar com o Nodo;
- *Ativo*: indica que a porta desse serviço será utilizada para computar a quantidade de serviços acessados pelo Gestor ao host do Nodo. Tal quantidade de serviços é posteriormente usada como a entrada de serviços não acessados para o Sistema de Inferência de Rede (SIR);
- *Inativo*: indica que a porta desse serviço não será utilizada para computar a quantidade de serviços acessados pelo Gestor ao host do Nodo.

sistema = (codigo: TINYTEXT, nome: TEXT, status: TEXT, data_inicio: DATE, data_fim: DATE).

Base de dados Inebulosa:

antecedente = (cod_regra: INT, cod_termo: INT);

regra = (cod_regra: INT, cod_termo_cons: INT, cod_sistema: INT);

segmento = (cod_termo: INT, cod_segmento: INT, intervalo_esquerdo: DOUBLE, intervalo_direito: DOUBLE, coef_angular: DOUBLE, coef_linear: DOUBLE, mi_esquerdo: DOUBLE, mi_direito: DOUBLE);

sistema = (cod_sistema: INT, nom_sistema: VARCHAR);

termo = (cod_termo: INT, cod_variavel: INT, nom_termo: VARCHAR);

variavel_linguistica = (cod_variavel: INT, nom_variavel: VARCHAR, ini_universo: INT, fim_universo: INT, cod_sistema: INT, flag_antecedente: CHAR).

4.3 Especificação dos Métodos do Gestor de Software na Nuvem

O programa Gestor de *Software* na Nuvem se encontra implementado em cinco pacotes: `interface_usuario`, `interface_rede`, `controle`, `DAO` e `modelo`. A seguir, são comentadas todas as classes desses pacotes e as Tabelas de números 3 a 25 apresentam uma descrição breve dos principais métodos de todas essas classes.

Pacote `interface_usuario`

Neste pacote se encontram implementadas as classes `FormularioPrincipal.java`, `InterfaceManterAtividade.java`, `InterfaceManterEquipe.java`, `IntefaceManterFase.java`, `InterfaceManterHost.java`, `InterfaceManterMonitoramento.java`, `InterfaceManterServico.java` e `InterfaceManterSistema.java`. As responsabilidades dessas classes são descritas a seguir:

- *FormularioPrincipal.java* → Esta classe processa a visualização da interface do Gestor de *Software* na Nuvem.
- *InterfaceManterAtividade.java* → Esta classe é responsável por manter a interface da Tabela Atividade.
- *InterfaceManterEquipe.java* → Esta classe é responsável por manter a interface da Tabela Equipe.

- *InterfaceManterFase.java* → Esta classe é responsável por manter a interface da Tabela Fase.
- *InterfaceManterHost.java* → Esta classe é responsável por manter a interface da Tabela Host.
- *InterfaceManterMonitoramento.java* → Esta classe é responsável por manter a interface da Tabela Monitoramento.
- *InterfaceManterServico.java* → Esta classe é responsável por manter a interface da Tabela Serviço.
- *InterfaceManterSistema.java* → Esta classe é responsável por manter a interface da Tabela Sistema.

As próximas tabelas mostram a descrição dos métodos mais importantes de algumas classes deste pacote:

Tabela 4 - Métodos da classe InterfaceManterAtividade.java.

Nome do Método	Descrição
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void preencheRegistrosAntigos (Atividade[] a, JTable jTable)	Este método preenche um vetor de objetos atividade com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.
void preencheTabelaAtividade(Atividade[] a, JTable jTable)	Preenche a tabela da interface do formulário principal com os dados da tabela Atividade do banco.
int quantRegistrosAntigos (JTable jTable)	Este método retorna a quantidade de registros já existentes na tabela Atividade mostrados na interface do formulário.

Tabela 5 - Métodos da classe InterfaceManterEquipe.java.

Nome do Método	Descrição
criaLinhaTabelaEquipe (JTable)	Método responsável por criar uma linha nova em branco na tabela da interface do formulário principal.
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void getRegistroSelecionado (Equipe e, JTable jTable)	Este método preenche um objeto equipe com os dados da linha selecionada na tabela da interface do formulário principal.

void preencheRegistrosAntigos (Equipe[] e, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.
void preencheRegistrosNovos (Equipe[] e, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros novos.
void preencheTabelaEquipe(Equipe[] e, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
int quantRegistrosAntigos (JTable jTable)	Este método retorna a quantidade de registros já existentes na tabela Atividade mostrados na interface do formulário.
int quantRegistrosNovos (JTable jTable)	Este método retorna a quantidade de registros novos na tabela Equipe mostrados na interface do formulário.

Tabela 6 - Métodos da classe InterfaceManterFase.java.

Nome do Método	Descrição
void criaLinhaTabelaFase (JTable jTable)	Método responsável por criar uma linha nova em branco na tabela da interface do formulário principal.
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void getRegistroSelecionado (Fase f, JTable jTable)	Este método preenche um objeto fase com os dados da linha selecionada na tabela da interface do formulário principal.
void preencheRegistrosAntigos (Fase[] f, JTable jTable)	Este método preenche um vetor de objetos fase com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.
void preencheRegistrosNovos (Fase[] f, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros novos.
void preencheTabelaFase (Fase[] f, JTable jTable)	Este método preenche um vetor de objetos fase com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
int quantRegistrosAntigos (JTable jTable)	Este método retorna a quantidade de registros já existentes na tabela Atividade mostrados na interface do formulário.
int quantRegistrosNovos (JTable jTable)	Este método retorna a quantidade de registros novos na tabela Fase mostrados na interface do formulário.

Tabela 7 - Métodos da classe InterfaceManterHost.java.

Nome do Método	Descrição
void criaLinhaTabelaHost (JTable jTable)	Método responsável por criar uma linha nova em branco na tabela da interface do formulário principal.
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void getRegistroSelecionado (Host h, JTable jTable)	Este método preenche um objeto host com os dados da linha selecionada na tabela da interface do formulário principal.
void preencheRegistrosAntigos (Host[] h, JTable jTable)	Este método preenche um vetor de objetos atividade com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.

boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void preencheTabelaHost(Host[] h, JTable jTable)	Este método preenche um vetor de objetos host com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
int quantRegistrosAntigos (JTable jTable)	Este método retorna a quantidade de registros já existentes na tabela host mostrados na interface do formulário.
int quantRegistrosNovos (JTable jTable)	Este método retorna a quantidade de novos registros existentes na tabela Host mostrados na interface do formulário.

Tabela 8 - Métodos da classe InterfaceManterMonitoramento.java.

Nome do Método	Descrição
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void getRegistroSelecionado (Monitoramento m, JTable jTable)	Este método preenche um objeto monitoracao com os dados da linha selecionada na tabela da interface do formulário principal.
void preencheTabelaMonitoramento(Monitoramento[] m, JTable jTable)	Este método preenche um vetor de objetos monitoramento com todas as informações alteradas ou não da tabela da interface do formulário dos registros.

Tabela 9 - Métodos da classe InterfaceManterServico.java.

Nome do Método	Descrição
void criaLinhaTabelaServico (JTable jTable)	Método responsável por criar uma linha nova em branco na tabela da interface do formulário principal.
boolean existeLinhaSelecionada (JTable jTable)	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
void getRegistroSelecionado (Servico serv, JTable jTable)	Este método preenche um objeto servico com os dados da linha selecionada na tabela da interface do formulário principal.
void preencheRegistrosAntigos (Servico[] serv, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.
void preencheRegistrosNovos (Servico[] serv, JTable jTable)	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
void preencheTabelaServico (Servico[] serv, JTable jTable)	Este método preenche um vetor de objetos serviço com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
int quantRegistrosAntigos (JTable jTable)	Este método retorna a quantidade de registros já existentes na tabela Atividade mostrados na interface do formulário.
int quantRegistrosNovos (JTable jTable)	Este método retorna a quantidade de novos existentes na tabela Serviços mostrados na interface do formulário.

Tabela 10 - Métodos da classe `InterfaceManterSistema.java`.

Nome do Método	Descrição
<code>void criaLinhaTabelaSistema (JTable jTable)</code>	Método responsável por criar uma linha nova em branco na tabela da interface do formulário principal.
<code>boolean existeLinhaSelecionada (JTable jTable)</code>	Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela da interface do formulário principal.
<code>void getRegistroSelecionado (Sistema s, JTable jTable)</code>	Este método preenche um objeto sistema com os dados da linha selecionada na tabela da interface do formulário principal.
<code>void preencheRegistrosAntigos (Sistema[] s, JTable jTable)</code>	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros antigos.
<code>void preencheRegistrosNovos (Sistema[] s, JTable jTable)</code>	Este método preenche um vetor de objetos equipe com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
<code>void preencheTabelaSistema(Sistema[] s, JTable jTable)</code>	Este método preenche um vetor de objetos sistema com todas as informações alteradas ou não da tabela da interface do formulário dos registros.
<code>int quantRegistrosAntigos (JTable jTable)</code>	Este método retorna a quantidade de registros já existentes na tabela Atividade mostrados na interface do formulário.
<code>int quantRegistrosNovos (JTable jTable)</code>	Este método retorna a quantidade de registros novos na tabela Sistema mostrados na interface do formulário.

Pacote `interface_rede`

Neste pacote encontram-se implementadas as classes *Cliente.java*, *Protocolo.java*, *Servidor.java* e *ServidorThread.java*. As responsabilidades dessas classes são descritas a seguir:

- *Cliente.java* → Esta classe é responsável pela iniciativa da comunicação por parte do Gestor para fazer a monitoração e a calibração.
- *Protocolo.java* → Esta classe é responsável por executar o protocolo de rede.
- *Servidor.java* → Esta classe é responsável por aguardar a conexão de um nodo.
- *ServidorThread.java* → Esta classe é responsável pelo processo servidor que atende a um nodo.

As próximas tabelas mostram a descrição dos métodos mais importantes de algumas classes deste pacote:

Tabela 11 - Métodos da classe *Cliente.java*.

Nome do Método	Descrição
int getTentativas (String IPhost)	Envia um ping para o host nodo específico várias vezes até obter resposta, retornando o total destas tentativas até o máximo de 10.
void executaCalibracao()	Este método mede o atraso e a quantidade de serviços ativos para todos os nodos que estão cadastrados como Fase no banco de dados.
double getAtraso (String IPhost, int portaHost)	Método que calcula o atraso de um nodo específico.
int getServicos (String IPhost, Servico[] servicos)	Verifica a quantidade de serviços de um nodo específico que estão com a porta aberta.

Tabela 12 - Métodos da classe *Protocolo.java*.

Nome do Método	Descrição
void executaIniciador()	Este método é responsável pelo processamento do lado iniciador do protocolo.
void executaRespondedor()	Este método é responsável pelo processamento do lado respondedor do protocolo.

Tabela 13 - Métodos da classe *Servidor.java*.

Nome do Método	Descrição
void iniciar ()	Método responsável por aguardar a conexão com o nodo, criando um processo servidor para atender a cada nodo.

Tabela 14 - Métodos da classe *ServidorThread.java*.

Nome do Método	Descrição
void recebeArtefato()	Método responsável por receber o arquivo de artefato enviado pelo nodo.
void run()	Método principal que aciona o protocolo para dar início à recepção do arquivo do artefato.

Pacote controle

Neste pacote se encontram implementadas as classes *Conexao.java*, *GestorSoft.java*, *MonitorNodos.java* e *MonitorNodosThread.java*. As responsabilidades dessas classes são descritas a seguir:

- *Conexao.java* → Esta classe é responsável pela abertura de conexão com o banco de dados.
- *GestorSoft.java* → Esta classe é responsável pelo acionamento das funcionalidades do Gestor, promovendo a interação entre os pacotes de interface e o pacote modelo.

- *MonitorNodos.java* → Esta classe é responsável pelo controle da monitoração dos nodos.
- *MonitorNodosThread.java* → Esta classe é responsável pelo processo de monitoramento dos nodos que executa de forma concorrente com o resto do Gestor.

As próximas tabelas mostram a descrição dos métodos mais importantes de algumas classes deste pacote:

Tabela 15 - Métodos da classe GestorSoft.java.

Nome do Método	Descrição
void alterarPorta (JLabel jLabelPorta)	Método responsável por obter um novo valor de porta do usuário e defini-lo para o Gestor. Em seguida, uma nova instância de servidor é criada com este novo valor de porta.
void atualizarBDAtividade (JTable jTable)	Método responsável por atualizar a tabela Atividade no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void atualizarBDEquipe (JTable jTable)	Método responsável por atualizar a tabela Equipe no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void atualizarBDFase (JTable jTable)	Método responsável por atualizar a tabela Fase no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void atualizarBDHost (JTable jTable)	Método responsável por atualizar a tabela Host no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void atualizarBDMonitoramento (JTable jTable)	Método responsável por abrir a caixa de diálogo informando que o usuário não pode incluir linhas na tabela Monitoramento.
void atualizarBDServico (JTable jTable)	Método responsável por atualizar a tabela Serviço no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void atualizarBDSistema (JTable jTable)	Método responsável por atualizar a tabela Sistema no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void calibrar()	Método responsável pela calibração dos nodos do <i>software</i> na nuvem. Tal método obtém os valores de atraso e serviços acessados de cada nodo.
void consultarAtividade (JTable jTable)	Método responsável por fazer o acesso à tabela Atividade do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void consultarEquipe (JTable jTable)	Método responsável por fazer o acesso à tabela Equipe do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void consultarFase (JTable jTable)	Método responsável por fazer o acesso à tabela Fase do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void consultarHost (JTable jTable)	Método responsável por fazer o acesso à tabela Host do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void consultarServico (JTable jTable)	Método responsável por fazer o acesso à tabela Serviço do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.

void consultarSistema (JTable jTable)	Método responsável por fazer o acesso à tabela Sistema do banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirAtividade (JTable jTable)	Método responsável por abrir a caixa de diálogo informando que o usuário não pode excluir linhas na tabela Atividade.
void excluirEquipe (JTable jTable)	Método responsável por excluir linha na tabela Equipe no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirFase (JTable jTable)	Método responsável por excluir linha na tabela Fase no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirHost (JTable jTable)	Método responsável por excluir linha na tabela Host no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirMonitoramento (JTable jTable)	Método responsável por excluir linha na tabela Monitoramento no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirServico (JTable jTable)	Método responsável por excluir linha na tabela Serviço no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void excluirSistema (JTable jTable)	Método responsável por excluir linha na tabela Sistema no banco de dados e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
void inserirLinhaAtividade (JTable jTable)	Método responsável por abrir a caixa de diálogo informando que o usuário não pode incluir linhas na tabela Atividade.
void inserirLinhaEquipe (JTable jTable)	Método responsável por inserir linha em branco na interface da tabela Equipe.
void inserirLinhaFase (JTable jTable)	Método responsável por inserir linha em branco na interface da tabela Fase.
void inserirLinhaHost (JTable jTable)	Método responsável por inserir linha em branco na interface da tabela <i>Host</i> .
void inserirLinhaMonitoramento (JTable jTable)	Método responsável por abrir a caixa de diálogo informando que o usuário não pode incluir linhas na tabela Monitoramento.
void inserirLinhaServico (JTable jTable)	Método responsável por inserir linha em branco na interface da tabela Serviço.
void inserirLinhaSistema (JTable jTable)	Método responsável por inserir uma linha em branco na interface da tabela Sistema.

Tabela 16 - Métodos da classe MonitorNodosThread.java.

Nome do Método	Descrição
double ajustaAtraso (double AtrasoAbsoluto, String IPhost)	Este método retorna o atraso relativo, ou seja, o atraso medido do nodo comparado com o anterior que foi obtido pela calibração o qual se encontra cadastrado no banco.
int ajustaServicos (int ServicosAcessados, String IPhost)	Este método calcula a quantidade de serviços sem acesso.
int artefatosAtrasadosRelativo (String IPhost)	Este método calcula a quantidade relativa de artefatos atrasados, ou seja, o percentual de artefatos atrasados em relação ao total de artefatos sob análise.
int artefatosErradosRelativo (String IPhost)	Este método calcular a quantidade relativa de artefatos errados.

int getPortaHostFase (Servico[] servicos, String IPhost)	Este método retorna o número da porta de um dado host que esteja funcionando como Fase, ou seja, um Nodo.
void iniciarMonitoramento ()	Método responsável por iniciar o monitoramento dos serviços.

Pacote DAO

Neste pacote se encontram implementadas as classes *AtividadeDAO.java*, *EquipeDAO.java*, *FaseDAO.java*, *HostDAO.java*, *MonitoramentoDAO.java*, *ServicoDAO.java* e *SistemaDAO.java*. As responsabilidades dessas classes são descritas a seguir:

- *AtividadeDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Atividade.
- *EquipeDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da Tabela Equipe.
- *FaseDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Fase.
- *HostDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Host.
- *MonitoramentoDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Monitoramento.
- *ServicoDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Servico.
- *SistemaDAO.java* → Esta classe busca os dados no banco de dados e atualiza a interface da tabela Sistema.

Tabela 17 - Métodos da classe AtividadeDAO.java.

Nome do Método	Descrição
void remove(Atividade atividade)	Método que exclui um registro da tabela Atividade do banco.
Atividade[] retrieveAll()	Método que consulta todos os registros da tabela Atividade do banco.
Atividade[] retrieveByHost (String host)	Método que consulta todos os registros da tabela Atividade do banco referentes a um host específico.
void save(Atividade atividade)	Método que insere um registro na tabela Atividade do banco.
void update(Atividade atividade)	Método que atualiza um registro da tabela Atividade do banco.

Tabela 18 - Métodos da classe EquipeDAO.java.

Nome do Método	Descrição
void remove(Equipe equipe)	Método que exclui um registro da tabela Equipe do banco.
Equipe[] retrieveAll()	Método que consulta todos os registros da tabela Equipe do banco.
void save(Equipe equipe)	Método que insere um registro na tabela Equipe do banco.
void update(Equipe equipe)	Método que atualiza um registro da tabela Equipe do banco.

Tabela 19 - Métodos da classe FaseDAO.java.

Nome do Método	Descrição
void remove(Fase fase)	Método que exclui um registro da tabela Fase do banco.
Fase[] retrieveAll()	Método que consulta todos os registros da tabela Fase do banco.
Fase retrieveByHost (String host)	Método que consulta todos os registros da tabela Fase do banco referentes a um host específico.
void save(Fase fase)	Método que insere um registro na tabela Fase do banco.
void update(Fase fase)	Método que atualiza um registro da tabela Fase do banco.

Tabela 20 - Métodos da classe HostDAO.java.

Nome do Método	Descrição
void remove(Host host)	Método que exclui um registro da tabela Host do banco.
Host[] retrieveAll()	Método que consulta todos os registros da tabela Host do banco.
Host retrieveByIP (String IP)	Método que consulta todos os registros da tabela Host do banco referentes a um host específico.
void save(Host host)	Método que insere um registro na tabela Host do banco.
void update(Host host)	Método que atualiza um registro da tabela Host do banco.

Tabela 21 - Métodos da classe MonitoramentoDAO.java.

Nome do Método	Descrição
void remove(Monitoramento monitoramento)	Método que exclui um registro da tabela Monitoramento do banco.
Monitoramento[] retrieveAll()	Método que consulta todos os registros da tabela Monitoramento do banco.
void save(Monitoramento monitoramento)	Método que insere um registro na tabela Monitoramento do banco.
void update(Monitoramento monitoramento)	Método que atualiza um registro da tabela Monitoramento do banco.

Tabela 22 - Métodos da classe *ServicoDAO.java*.

Nome do Método	Descrição
void remove(Servico servico)	Método que exclui um registro da tabela Servico do banco.
Servico[] retrieveAll()	Método que consulta todos os registros da tabela Servico do banco.
void save (Servico servico)	Método que insere um registro na tabela Servico do banco.
void update (Servico servico)	Método que atualiza um registro da tabela Servico do banco.

Tabela 23 - Métodos da classe *SistemaDAO.java*.

Nome do Método	Descrição
void remove(Sistema sistema)	Método que exclui um registro da tabela Sistema do banco.
Sistema[] retrieveAll()	Método que consulta todos os registros da tabela Sistema do banco.
void save(Sistema sistema)	Método que insere um registro na tabela Sistema do banco.
void update(Sistema sistema)	Método que atualiza um registro da tabela Sistema do banco.

Pacote modelo

Neste pacote encontram-se implementadas as classes *Atividade.java*, *CalculaCoeficientes.java*, *DeFuzzy.java*, *DeFuzzyBean.java*, *Equipe.java*, *Fase.java*, *Host.java*, *Monitoramento.java*, *Servico.java* e *Sistema.java*. As responsabilidades dessas classes são descritas a seguir:

- *Atividade.java* → Esta classe é responsável pela definição do objeto da entidade Atividade.
- *CalculaCoeficientes.java* → Esta classe é responsável pelo cálculo dos coeficientes angular e linear das equações dos segmentos de reta das funções de pertinência dos termos das variáveis linguísticas dos sistemas de inferências nebulosos.
- *DeFuzzy.java* → Esta classe faz a inferência nebulosa, utilizando o método de defuzzyficação da média aritmética ponderada dos centroides pela área.
- *DeFuzzyBean.java* → Esta classe é responsável pela definição do objeto da entidade de inferência nebulosa defuzzy.
- *Equipe.java* → Esta classe é responsável pela definição do objeto da entidade Equipe.

- *Fase.java* → Esta classe é responsável pela definição do objeto da entidade Fase.
- *Host.java* → Esta classe é responsável pela definição do objeto da entidade Host.
- *Monitoramento.java* → Esta classe é responsável pela definição do objeto da entidade Monitoramento.
- *Servico.java* → Esta classe é responsável pela definição do objeto da entidade Serviço.
- *Sistema.java* → Esta classe é responsável pela definição do objeto da entidade Sistema.

4.4 Algoritmo de Inferência Nebuloso

O algoritmo de inferência nebuloso adotado segue o *Modelo de Larsen* (LARSEN 1981). Tal modelo foi escolhido por permitir o uso de qualquer função de pertinência nos termos consequentes das regras de inferência nebulosas e por ser mais fácil de se implementar, comparado com o modelo de Mamdani (MAMDANI, 1974) que é semelhante. Os modelos de Larsen e Mamdani fornecem praticamente os mesmos resultados, como poderá ser constatado nos testes apresentados no Capítulo 5.

A classe *DeFuzzy.java*, a qual contém a implementação do algoritmo de inferência nebuloso, utiliza o método de *defuzzyficação* da média aritmética ponderada dos centroides pela área. O pseudocódigo desse algoritmo é apresentado adiante.

```

INICIO DA INFERÊNCIA (entrada1, entrada2, entrada3, sistema)
  somatorioCentroide = 0, somatorioArea = 0;
  centroideRegra = 0, areaRegra = 0;
  SELECIONAR as Regras para o Sistema = sistema; // SIR ou SIS
  PARA cada Regra FAZER
    centroideRegra = 0, areaRegra = 0;
    Obter o 1º Termo da Regra
    p1 = Avaliar a pertinência do 1º Termo para entrada1;
    Obter o 2º Termo da Regra;
    p2 = Avaliar a pertinência do 2º Termo para entrada2;
    Obter o 3º Termo da Regra;
    p3 = Avaliar a pertinência do 3º Termo para entrada3;
    grau de ativação = menor valor entre p1, p2 e p3;
    SE grau de ativação > 0 ENTÃO
      Obter termoConsequente da Regra
      tConsAlt = Multiplicar a pertinência do termoConsequente
                  pelo grau de ativação
      CalcularCentroideRegra (tConsAlt, centroideRegra, areaRegra)
      somatorioCentroide = somatorioCentroide
                          + (centroideRegra * areaRegra);
      somatorioArea = somatorioArea + areaRegra;
    FIM SE
  FIM PARA

```

ValorFinal = somatorioCentroide / somatorioArea;
FIM INFERÊNCIA

O algoritmo de inferência, em pseudocódigo, mostrado anteriormente, recebe como argumentos o valor das três entradas do sistema em execução (SIR ou SIS), em seguida são calculadas, para todas as regras nebulosas, as pertinências dessas entradas avaliadas pelas respectivas funções dos termos presentes em cada regra.

No caso da menor pertinência ser diferente de zero, o algoritmo passa para o cálculo do centroide, da área e do produto área × centroide do termo consequente da regra computada multiplicado pelo grau de ativação (valor da menor pertinência das entradas). Esses valores calculados são acumulados, para depois de processar todas as regras para as entradas recebidas, encerrar o algoritmo, computando a média aritmética dos centroides dos termos consequentes alterados pela menor pertinência das regras ativadas. Tal média é o valor inferido pelo sistema nebuloso executado.

4.5 Algoritmo de Monitoramento do Gestor de *Software* na Nuvem

O algoritmo de monitoramento do Gestor de *Software* na Nuvem é responsável pela observação periódica do funcionamento dos nodos que processam cada Fase do produto de *software* que se encontra em desenvolvimento. Tal algoritmo faz a verificação do funcionamento dos Nodos em relação à sua funcionalidade de rede e em relação à questão da segurança, procurando identificar se o Nodo consultado encontra-se em perfeito funcionamento.

O monitoramento processa a análise de cada Nodo por meio da execução dos sistemas de inferência nebulosos implementados (SIR e SIS). Cada um desses sistemas de inferência nebulosos é executado após a computação dos valores das suas entradas. O SIR tem como entradas o atraso de acesso ao Nodo, a quantidade de serviços sem acesso ao Nodo e o número de tentativas de comunicação com o Nodo.

A entrada atraso para o SIR consiste em medir o tempo de resposta em milissegundos do Nodo para o Gestor, todavia tal atraso medido durante o monitoramento de forma absoluta não é indicativo de funcionamento normal ou com problema, visto que não se conhece, a priori, o tempo de processamento para uma mensagem fluir do Gestor a um Nodo específico. Sendo assim, o que importa é saber o quanto o atraso medido durante o monitoramento

encontra-se afastado de um atraso normal da comunicação entre o Gestor e um Nodo. Tal atraso medido durante o monitoramento deve ser, portanto, comparado com outro atraso obtido anteriormente o qual seria indicativo de funcionamento normal (valor de referência). Essa medida do atraso de referência é feita pela funcionalidade de calibração.

A calibração é responsável por medir o atraso de referência na comunicação entre o Gestor e um Nodo, assim como a verificação dos serviços em funcionamento nos Nodos. Esses dois valores de referência (atraso e quantidade de serviços com acesso) são armazenados na tabela host do banco de dados do Gestor de *Software* na Nuvem. Conclui-se que a calibração deve ser processada antes da execução do monitoramento.

Resumindo, durante o monitoramento, para o processamento do SIR são realizadas três medidas pelo Gestor em relação ao Nodo sob análise para se obter as suas três entradas, ou seja:

- Atraso Relativo → Trata-se do atraso ajustado obtido pela razão entre o atraso absoluto (medido durante o monitoramento) e o atraso da calibração. Caso essa razão supere o valor 10 ou caso haja algum problema de conexão que ocorra no momento da medida do atraso absoluto, será assumido o valor 10 para o atraso relativo, o qual corresponde ao valor máximo do universo de discurso da variável linguística atraso do SIR.
- Serviços sem Acesso → Trata-se da diferença entre os serviços com acesso medido no momento do monitoramento e os serviços acessados medidos durante a calibração. Destaca-se apenas que o valor dessa diferença deve ficar dentro dos limites do universo de discurso da variável linguística serviços sem acesso do SIR.
- Número de Tentativas → Trata-se da quantidade de vezes que o Gestor tenta se conectar com o Nodo até um limite máximo de 10. Caso haja um problema de conexão, esse limite máximo será alcançado e, em caso contrário, o Gestor deve obter resposta do Nodo com uma quantidade pequena de tentativas, ainda que haja congestionamento na rede.

Após a obtenção dessas três entradas, o SIR finalmente é processado pela classe *DeFuzzy*, a qual retorna o valor do índice de problema rede existente entre o Gestor e o Nodo

sob análise. Tal índice fica armazenado para ser fornecido como entrada para o processamento do SIS. O SIS possui, além dessa entrada, mais duas outras. a saber:

- Artefatos Fora do Prazo → Entende-se por Artefatos Fora do Prazo (AFP) os arquivos que foram enviados fora do prazo determinado pelo Gestor. As datas de início e fim podem ser visualizadas na tabela fase. Quando a data de fim é menor do que a data de entrega do artefato, este é considerado como Artefato Fora do Prazo.
- Artefatos Com Erro → Entende-se por Artefatos Com Erro (ACE) os arquivos que foram entregues com erro, como por exemplo, com ausência da informação esperada, ou quando não é possível abrir o arquivo.

Durante o monitoramento, o programa Gestor consulta diversas informações presentes no seu banco de dados, tanto da base gestsoftnuvem quanto da base Inebulosa. Todas as tabelas da base de dados Inebulosa são consultadas para executar o SIR e o SIS. Já no caso da base de dados gestsoftnuvem, as tabelas Atividade, Fase, Host e Servicos são consultadas, enquanto a tabela Monitoramento é atualizada com o resultado das inferências do SIR e do SIS. A Figura 28 a seguir apresenta o diagrama de atividades da UML com a lógica do algoritmo de monitoramento.

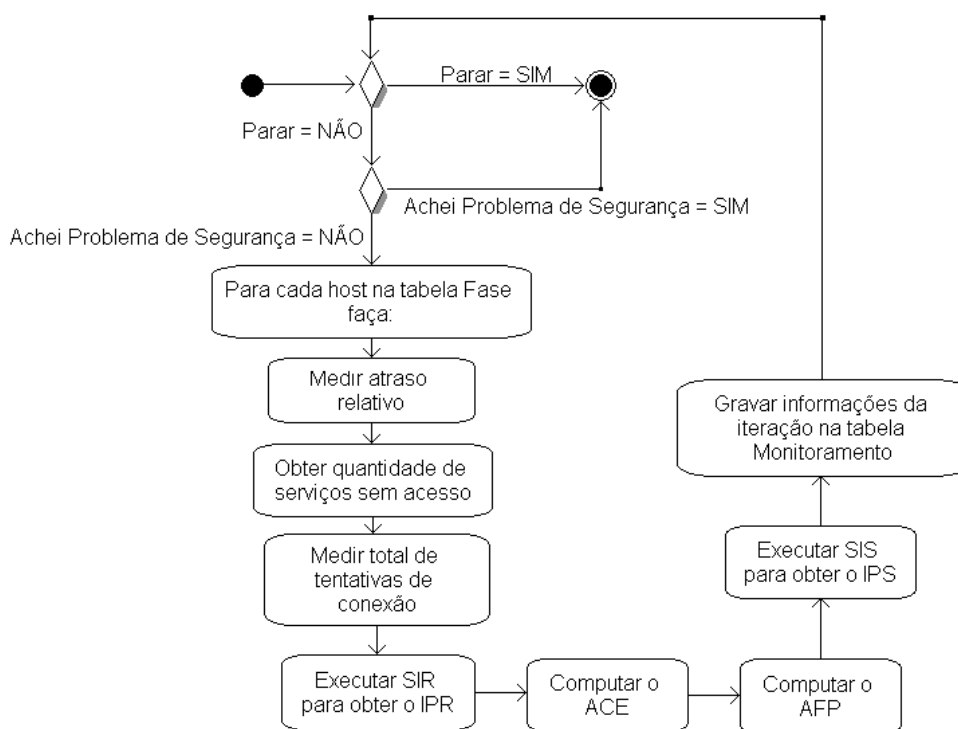


Figura 28 – Algoritmo de Monitoramento do Gestor de *Software* na Nuvem

A Figura 28 mostra que o algoritmo de monitoramento é executado até um comando de parada ser acionado pelo usuário ou até um problema de segurança em algum Nodo for detectado. Em cada iteração do algoritmo, são executados o SIR e o SIS para todos os nodos cadastrados na tabela Host. Os valores do IPR e do IPS inferidos de cada Nodo em cada iteração do algoritmo são armazenados na tabela Monitoramento, juntamente com o endereço IP do Nodo e com as entradas atraso relativo, serviços sem acesso, número de tentativas, total de artefatos com erro e total de artefatos fora do prazo.

4.6 Interfaces do Gestor de *Software* na Nuvem

Esta seção apresenta a visualização das telas da Interface do Gestor de *Software* na Nuvem e ainda uma descrição sobre as funções de cada item.

A tela inicial é apresentada na Figura 29. Ela possui os itens Gerenciar, Monitorar e Ajuda.

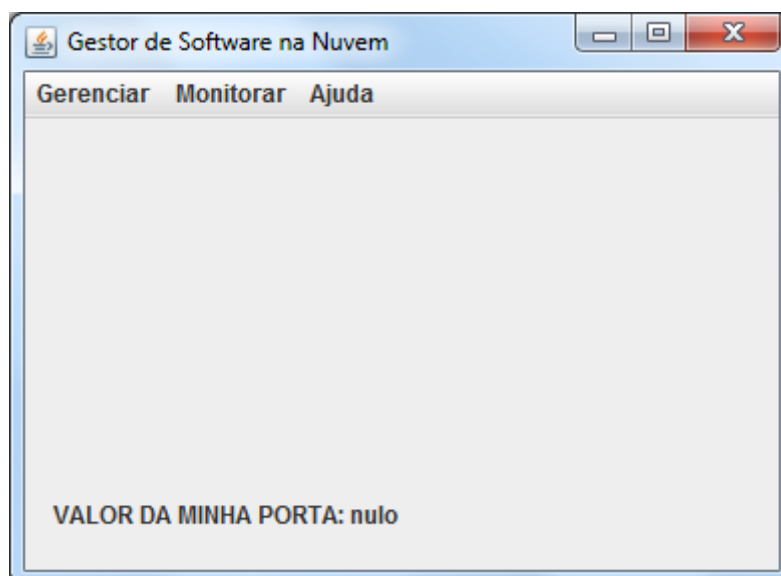


Figura 29 – Interface do Gestor de *Software* na Nuvem.

O menu Gerenciar apresenta as opções de Definir Minha Porta, Calibrar, Manter Banco de Dados e Sair.

A Figura 30 ilustra as opções do menu Gerenciar.

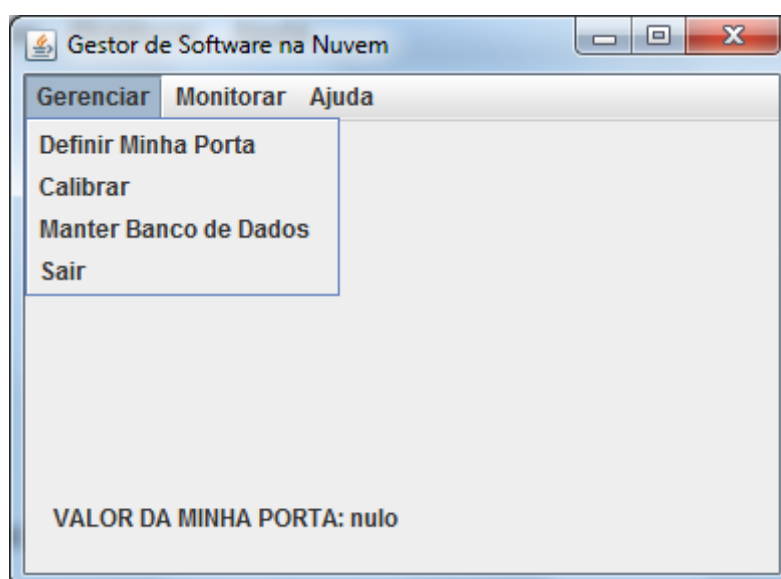


Figura 30 – Opções do menu Gerenciar.

Escolhendo a opção Definir Minha Porta, é apresentada a seguinte caixa de diálogo, como mostra a Figura 31. Nesta caixa, o usuário deverá definir o número da porta utilizada pelo Gestor.

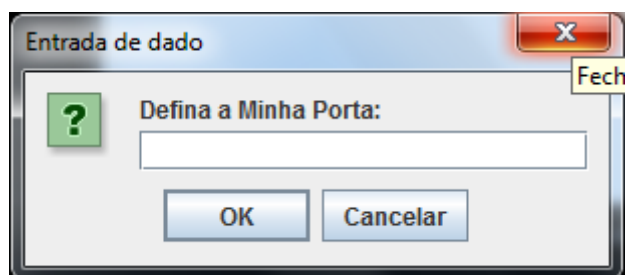


Figura 31 – Entrada de Dado: Defina a Minha Porta.

A opção Calibrar é executada sempre antes de começar a inferência dos sistemas. Esta função é responsável por ajustar o Gestor e os nós envolvidos. A Figura 32 mostra a interface da calibração.

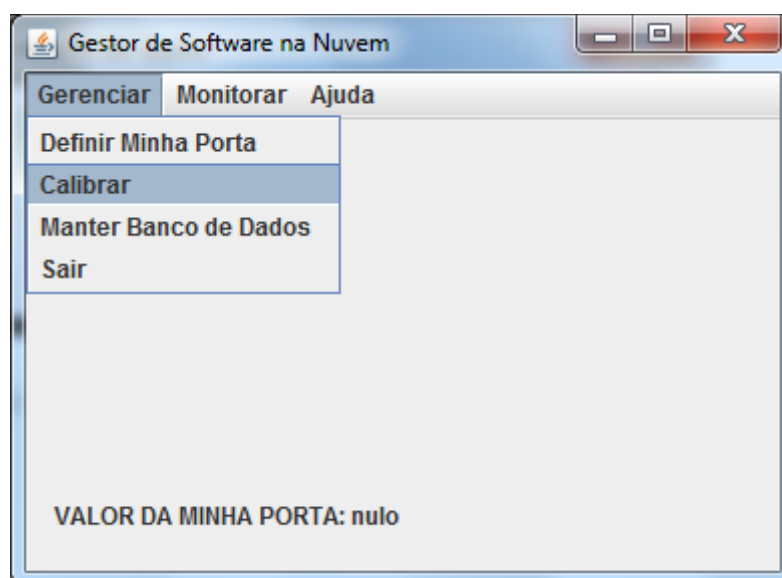


Figura 32 – Calibrar.

Após escolhida a opção Manter Banco de Dados, é aberta uma caixa onde é possível escolher a tabela a ser mantida, como mostra a Figura 33.

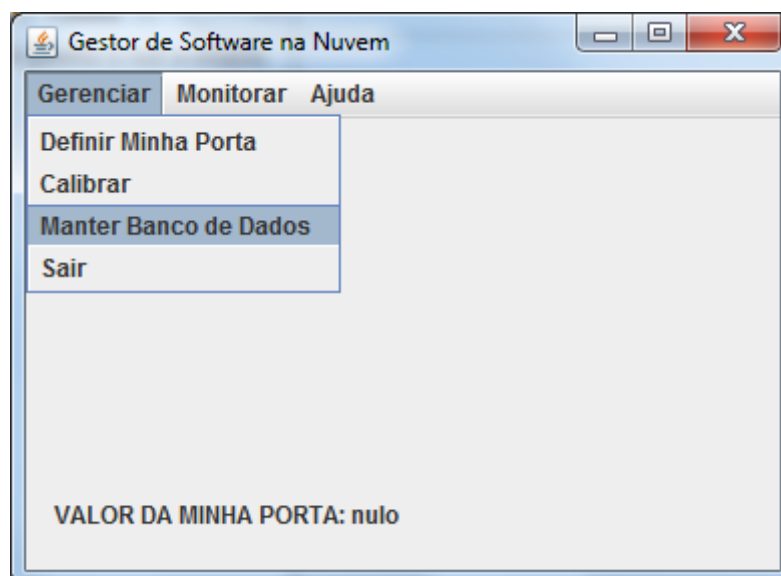


Figura 33 – Manter Banco de Dados.

Entre as opções de tabelas a serem mantidas estão: Sistema, Equipe, Fase, Atividade, Host, Serviço e Monitoramento. Após escolhida a tabela, é possível Inserir Nova Linha, Excluir Registro e Atualizar Banco de Dados para a tabela selecionada, conforme ilustrado na Figura 34.

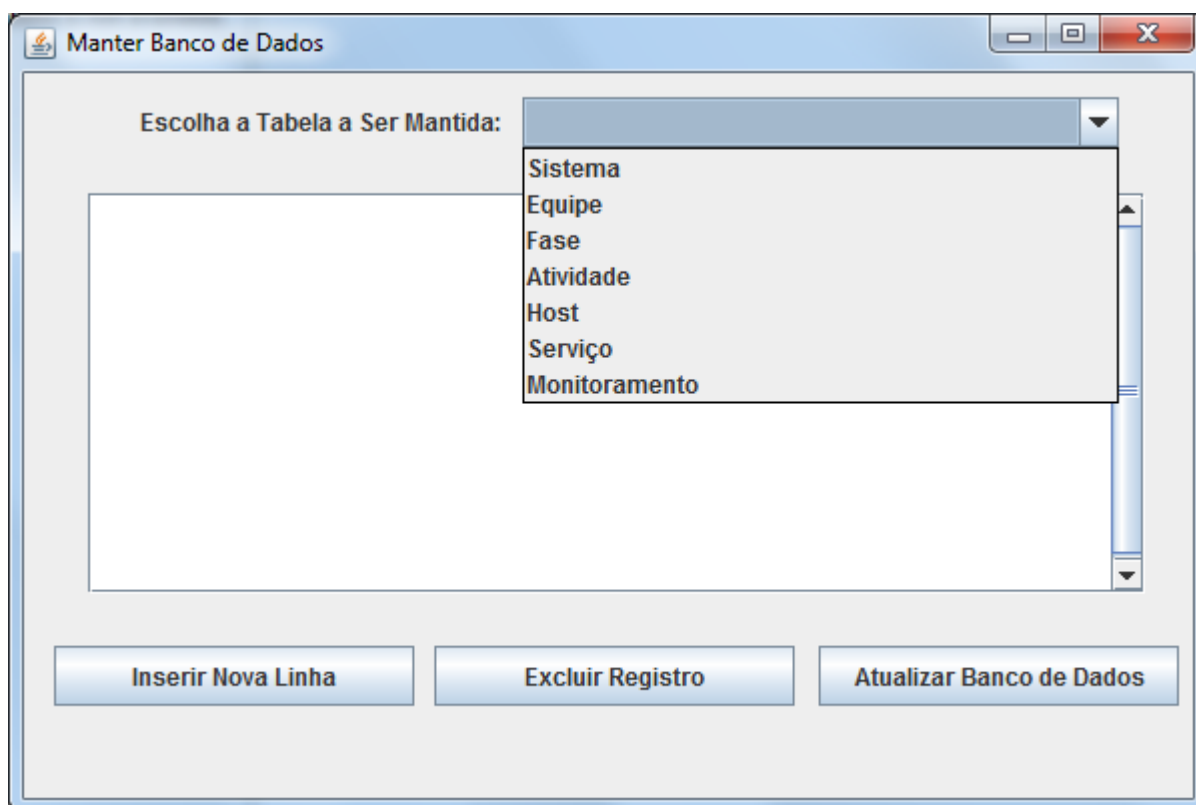
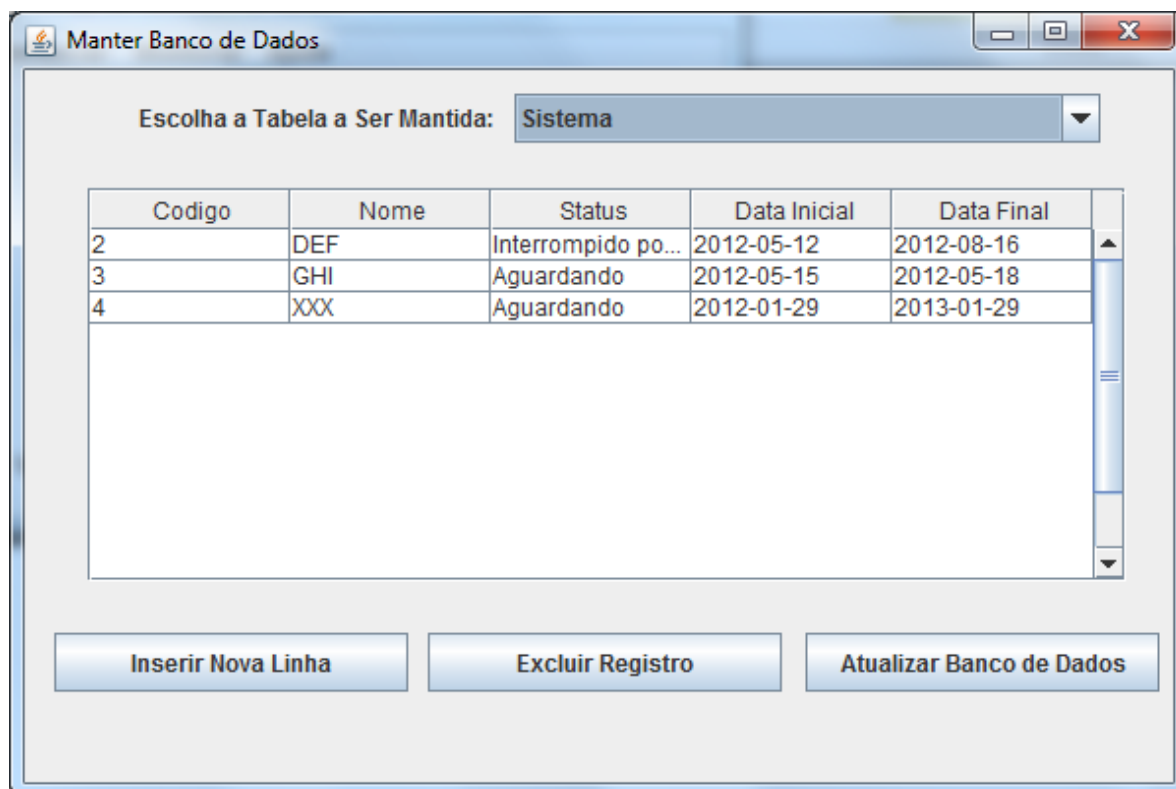


Figura 34 – Tabela a Ser Mantida.

A Figura 35 apresenta a Tabela Sistema como sendo a escolhida para ser mantida. Os campos pertencentes a esta opção são: Código, Nome, Status, Data Inicial e Data Final. O Código é a indicação da operação. O campo Nome indica a nome da Fase do Projeto. A Data Inicial apresenta o início da data do projeto. A Data Final, o término do mesmo.



Escolha a Tabela a Ser Mantida: Sistema

Codigo	Nome	Status	Data Inicial	Data Final
2	DEF	Interrompido po...	2012-05-12	2012-08-16
3	GHI	Aguardando	2012-05-15	2012-05-18
4	XXX	Aguardando	2012-01-29	2013-01-29

Inserir Nova Linha Excluir Registro Atualizar Banco de Dados

Figura 35 – Tabela a Ser Mantida: Sistema.

A Figura 36 apresenta a Tabela Equipe. Seus campos são Código do Funcionário, Nome do Funcionário, Cargo do Funcionário e Código da Fase. Seus campos são autoexplicativos.

Escolha a Tabela a Ser Mantida: Equipe

Código do Funcionário	Nome do Funcionário	Cargo do Funcionário	Código da Fase
2	Orlando Bernardo	Coordenador	2
3	João Araujo	Supervisor	0
4	Stella	Secretaria	0
7	Rafael	Boy	6
8	Leonardo	Contínuo	3
9	Carlos	Analista	1

Inserir Nova Linha Excluir Registro Atualizar Banco de Dados

Figura 36 – Tabela a Ser Mantida: Equipe.

A Figura 37 a seguir, apresenta a Tabela Fase. Nela, os campos Código, Nome, Host e Status são exibidos.

Escolha a Tabela a Ser Mantida: Fase

Código	Nome	Host	Status
1	Requisitos	192.168.137.166	Ativa
2	Lixo	192.168.137.159	Ativa

Inserir Nova Linha Excluir Registro Atualizar Banco de Dados

Figura 37 – Tabela a Ser Mantida: Fase.

A Figura 38 ilustra um exemplo onde a tabela a ser mantida escolhida foi a Atividade. Nesta tabela, são visualizados os campos Código, Host, Status e Arquivo do Artefato. O Código é a identificação numérica da operação. O Host é o endereço IP da máquina. O status pode ser classificado em Validando, Analisado, Correto, Atrasado, Errado e Atrasado e Errado. Estas definições foram descritas no Capítulo 4, Seção 4.2.

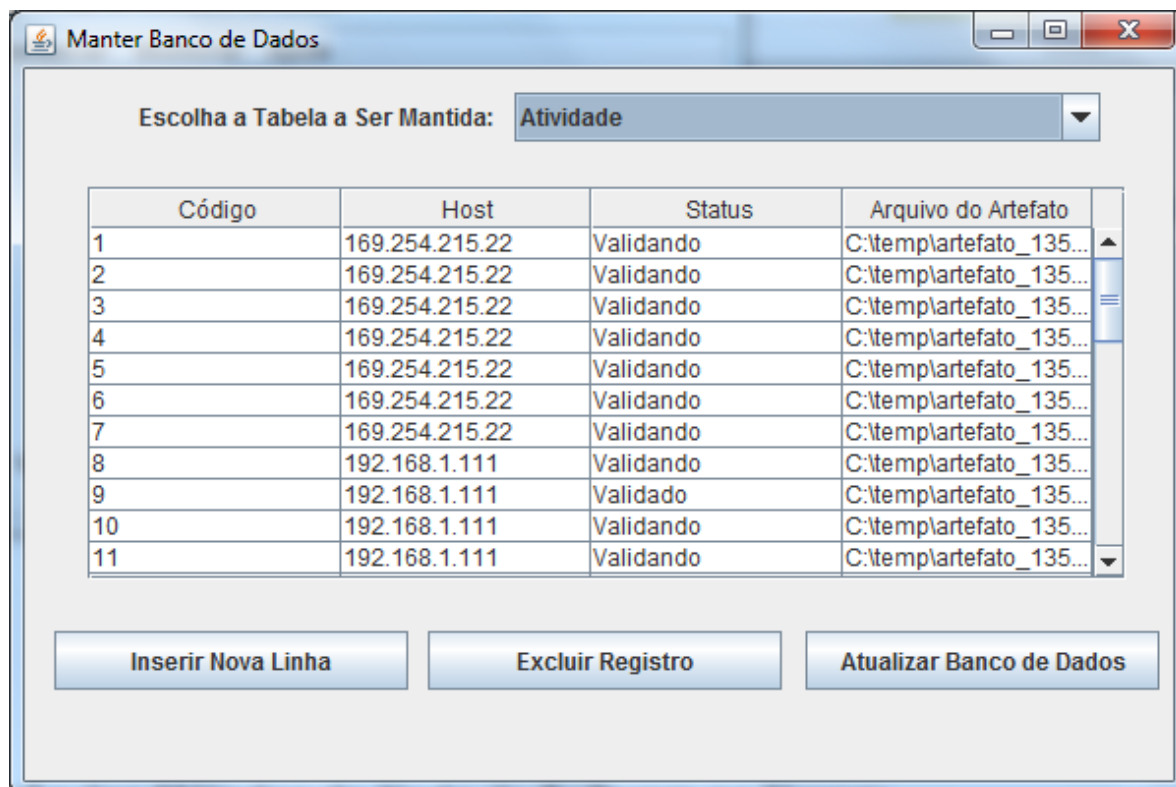


Figura 38 – Tabela a Ser Mantida: Atividade.

A Figura 39, a seguir, apresenta a Tabela Host. Seus campos são Endereço IP, Total de Serviços e Valor do Atraso.

The screenshot shows a window titled "Manter Banco de Dados" with a dropdown menu set to "Host". Below the menu is a table with three columns: "Endereço IP", "Total de Serviços", and "Valor do Atraso". The table contains two rows of data. At the bottom of the window are three buttons: "Inserir Nova Linha", "Excluir Registro", and "Atualizar Banco de Dados".

Endereço IP	Total de Serviços	Valor do Atraso
192.168.137.159	0	-1.0
192.168.137.166	0	-1.0

Figura 39 – Tabela a Ser Mantida: Host.

A Figura 40 apresenta a Tabela Serviço. Seus campos são: Código, Status, Host e Porta.

The screenshot shows a window titled "Manter Banco de Dados" with a dropdown menu set to "Serviço". Below the menu is a table with four columns: "Código", "Status", "Host", and "Porta". The table contains nine rows of data. At the bottom of the window are three buttons: "Inserir Nova Linha", "Excluir Registro", and "Atualizar Banco de Dados".

Código	Status	Host	Porta
1	Ativo	192.168.137.166	3306
2	Fase	192.168.137.166	4446
3	Ativo	192.168.137.166	8080
4	Ativo	192.168.137.166	4447
5	Ativo	192.168.137.166	4448
6	Ativo	192.168.137.159	4447
7	Fase	192.168.137.159	4446
8	Inativo	192.168.137.159	5555
9	Ativo	192.168.137.159	4449

Figura 40 – Tabela a Ser Mantida: Serviço.

A Figura 41 apresenta a Tabela Monitoramento. Seus campos são: Código, Host e Atraso.



Escolha a Tabela a Ser Mantida: **Monitoramento**

Código	Host	Atraso
35	192.168.1.103	0.72468778
36	192.168.1.103	0.16443938
37	192.168.1.103	1.0
38	192.168.1.103	1.0
39	192.168.1.103	1.0
40	192.168.1.103	1.0
41	192.168.1.103	1.0
42	192.168.1.103	0.15664946
43	192.168.1.103	1.0
44	192.168.1.103	0.12191103

Inserir Nova Linha **Excluir Registro** **Atualizar Banco de Dados**

Figura 41 – Tabela a Ser Mantida: Monitoramento.

A Figura 42 apresenta as opções do menu Monitorar: Iniciar Monitoramento e Cancelar Monitoramento, conforme mostrado abaixo.

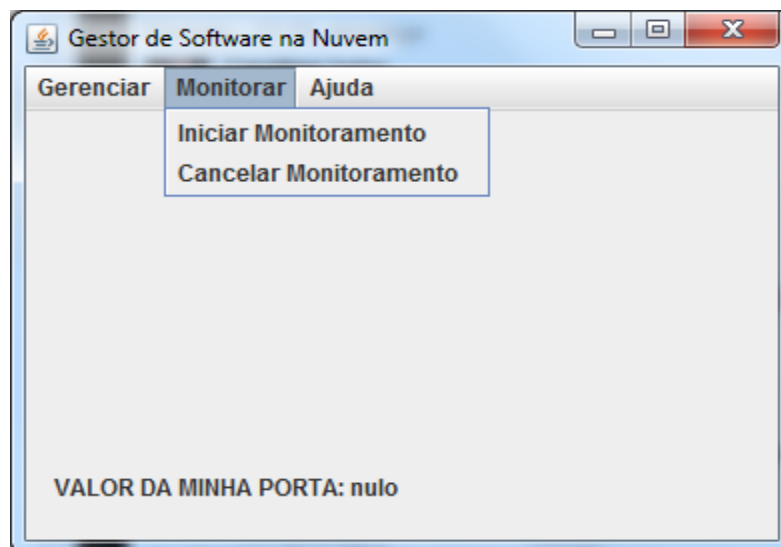


Figura 42 – Opções do campo Monitorar.

A Figura 43 apresenta um exemplo de monitoração do host 192.168.137.166, em sua primeira iteração.

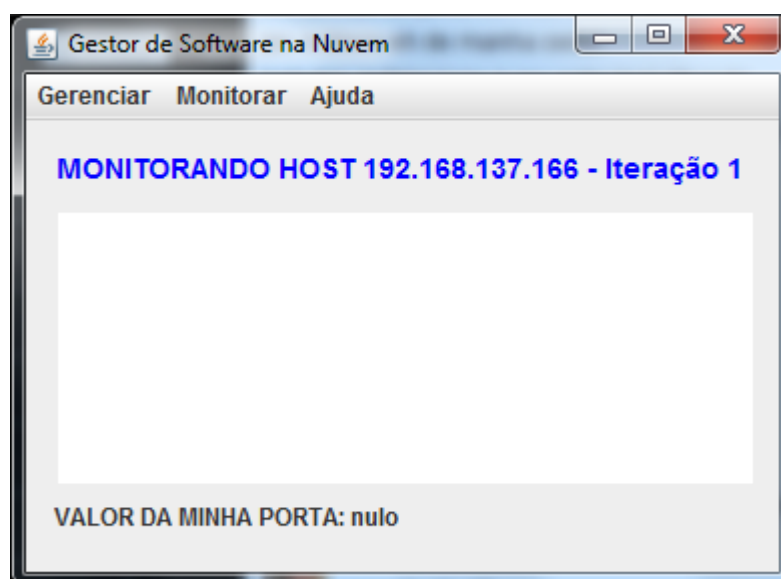


Figura 43 – Monitorar: Iniciar Monitoramento.

No exemplo apresentado na Figura 44, são mostrados os resultados da iteração 8, com a quantidade de Serviços sem acesso, Atraso, Tentativas, Índice de Problemas na Rede, Artefatos Atrasados, Artefatos com Erro e Índice de Problemas de Segurança.

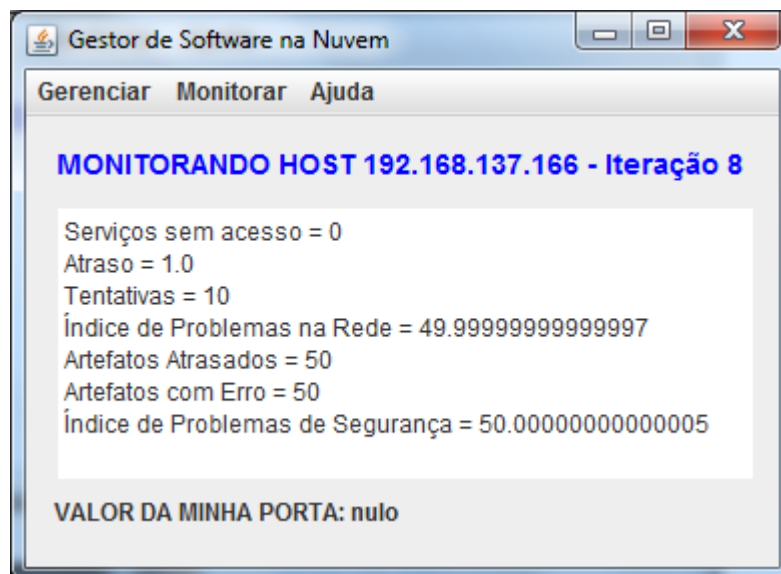


Figura 44 – Exemplo de Monitoração.

Caso o usuário deseje cancelar o monitoramento, é possível escolher a opção Cancelar Monitoramento, conforme mostrado na Figura 45.

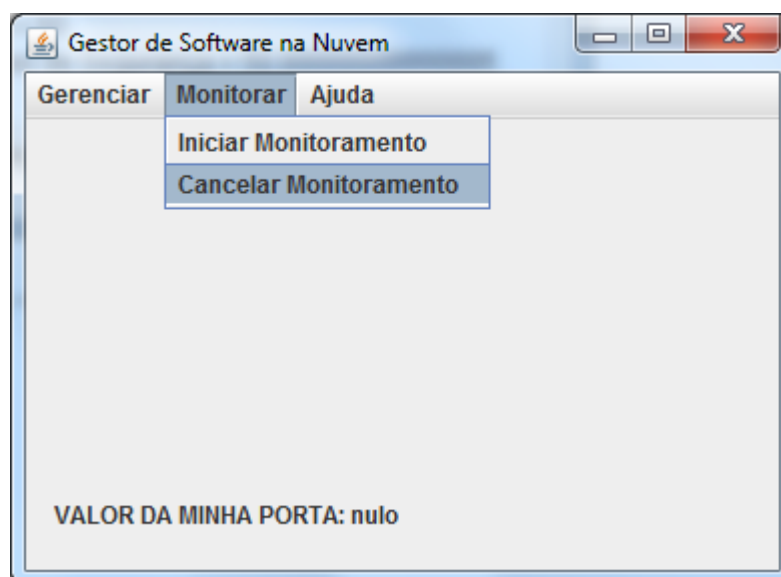


Figura 45 – Cancelar Monitoramento.

Após escolhida a opção de Cancelar o Monitoramento, é exibida a caixa de diálogo, informando a mensagem “Aguarde conclusão do monitoramento”, conforme apresentada na Figura 46.

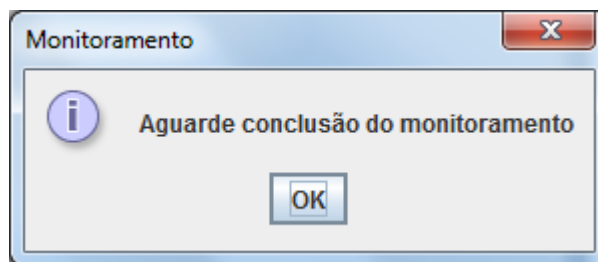


Figura 46 – Caixa de Diálogo do Monitoramento.

A Figura 47 apresenta o menu Ajuda, onde é possível selecionar a opção Sobre.

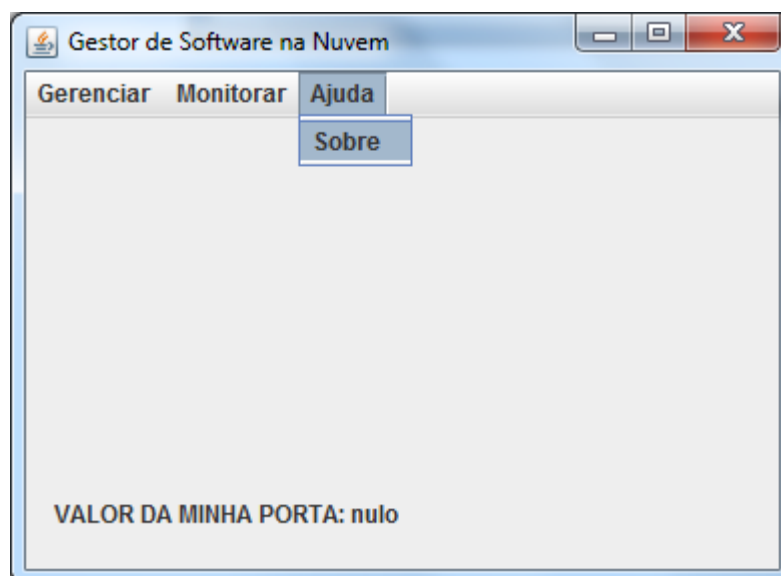


Figura 47 – Menu Ajuda: Sobre.

Ao selecionar a opção Sobre, é aberta outra caixa de diálogo, informando o nome do *Software* e a versão do mesmo, conforme apresentado na Figura 48.

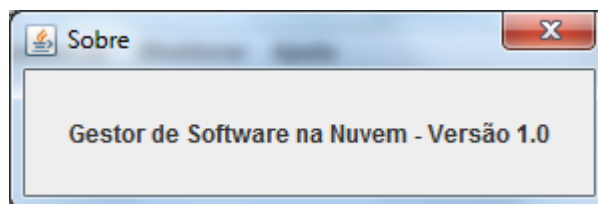


Figura 48 – Sobre: Gestor de *Software* na Nuvem – Versão 1.0.

4.7 Especificação dos Métodos do Nodo de Software na Nuvem

O programa Nodo de *Software* na Nuvem se encontra implementado em quatro pacotes: *interface_usuario*, *interface_rede*, *controle* e *modelo*. A seguir, são comentadas todas as classes desses pacotes e as Tabelas de números 24 a 27 apresentam uma descrição breve dos principais métodos de todas essas classes.

Pacote *interface_usuario*

Neste pacote encontra-se implementada a classe *FormularioPrincipal.java*. A responsabilidade dessa classe é descrita a seguir:

- *FormularioPrincipal.java* → Esta classe processa a visualização da interface do Nodo de Software na Nuvem.

Pacote *interface_rede*

Neste pacote encontram-se implementadas as classes *Cliente.java*, *Protocolo.java* e *Servidor.java*. A responsabilidade dessas classes é descrita a seguir:

- *Cliente.java* → Esta classe é responsável pela iniciativa da comunicação por parte do nodo para enviar um artefato para o Gestor.
- *Protocolo.java* → Esta classe é responsável por executar o protocolo de rede.
- *Servidor.java* → Esta classe é responsável por aguardar a conexão do Gestor.

Tabela 24 - Métodos da classe *Cliente.java*.

Nome do Método	Descrição
void transfereArtefato (Socket socketcliente, JLabel jLabelEnviaArt, JProgressBar jProgBarEnviaArt, String outFile)	Método responsável por enviar o arquivo do artefato propriamente dito.
void enviaArtefato(JLabel jLabelEnviaArt, JProgressBar jProgBarEnviaArt, String outFile)	Método responsável por fazer a negociação da transmissão com o Gestor para dar início ao envio do arquivo do artefato.

Tabela 25 - Métodos da classe *Protocolo.java*.

Nome do Método	Descrição
void executaRespondedor()	Este método é responsável pelo processamento do lado respondedor do protocolo.
void executaIniciador()	Este método é responsável pelo processamento do lado iniciador do protocolo.

Tabela 26 - Métodos da classe *Servidor.java*.

Nome do Método	Descrição
void run()	Método responsável por aguardar a conexão com o Gestor, executando o protocolo de rede para processar a solicitação recebida.

Pacote controle

Neste pacote encontra-se implementada a classe *ControlaNodo.java*. A responsabilidade dessa classe é descrita a seguir:

- *ControlaNodo.java* → Esta classe é responsável pelo acionamento das funcionalidades do nodo, promovendo a interação entre os pacotes de interface e o pacote modelo.

Tabela 27 - Métodos da classe *ControlaNodo.java*.

Nome do Método	Descrição
void alterarPorta (JLabel jLabelPorta, JLabel jLabelrecepcao)	Método responsável por obter um novo valor de porta do usuário e defini-lo para o nodo e em seguida, uma nova instância de servidor é criada com este novo valor de porta.
void alterarPortaGestor (JLabel jLabelIDgestor)	Método responsável por obter um novo valor de porta do usuário e defini-lo para o Gestor.
void alterarIPGestor (JLabel jLabelIDgestor)	Método responsável por obter um novo valor de IP do usuário e defini-lo para o Gestor

Pacote modelo

Neste pacote encontra-se implementada a classe *Gestor.java*. A responsabilidade dessa classe é descrita a seguir:

- *Gestor.java* → Esta classe é responsável pela definição do objeto da entidade *Gestor*.

4.8 Interfaces do Nodo de Software na Nuvem

Esta seção apresenta a visualização das telas da Interface do Nodo do *Software* na Nuvem e ainda uma descrição sobre as funções de cada item.

A tela inicial é apresentada na Figura 49. Ela possui os itens Arquivo, Identificação do Gestor e Ajuda.

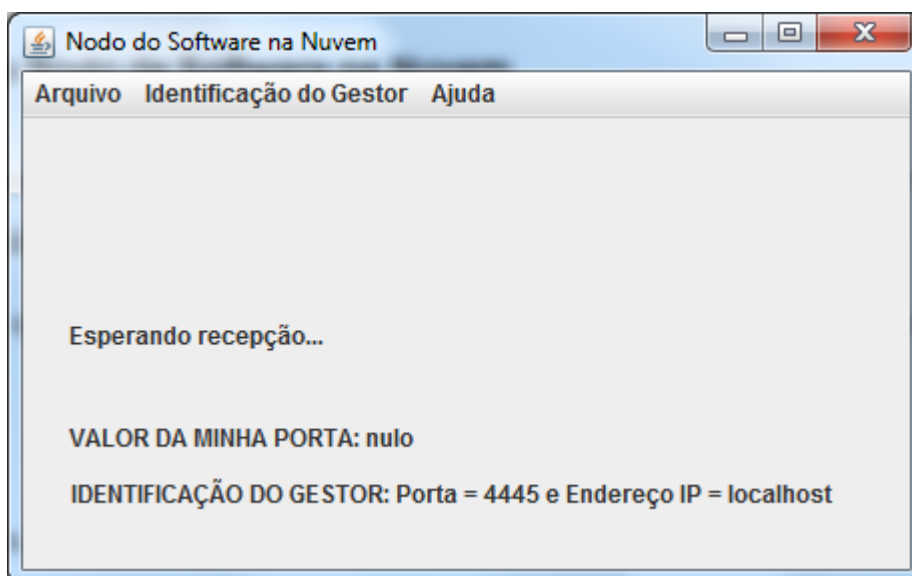


Figura 49 – Menus do *software* *Nodo do Software* na Nuvem.

O menu Arquivo possui as opções Definir Minha Porta, Enviar Artefato e Sair, conforme apresentado na Figura 50.

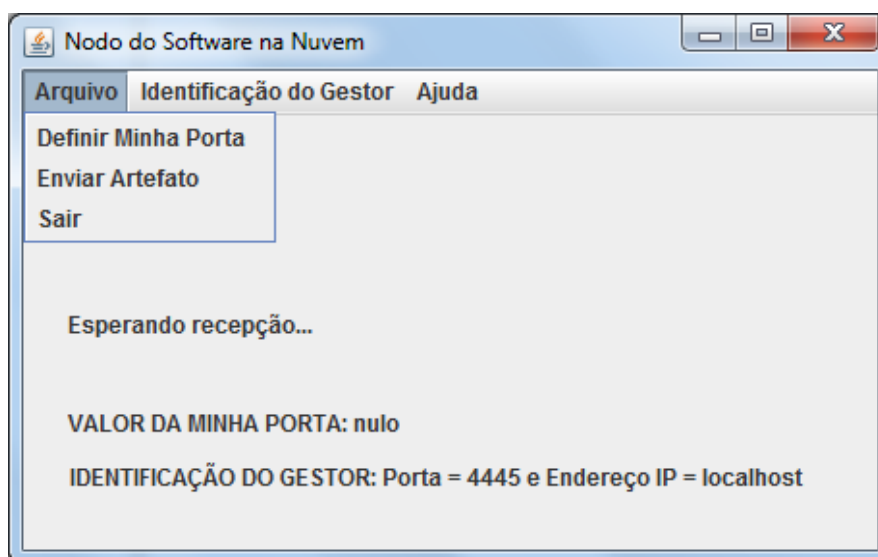


Figura 50 – Menus do *software* *Nodo do Software* na Nuvem.

Escolhendo a opção Definir Minha Porta, é apresentada a seguinte caixa de diálogo, como mostra a Figura 51. Nesta caixa, o usuário deverá definir o número da porta utilizada pelo Nodo.

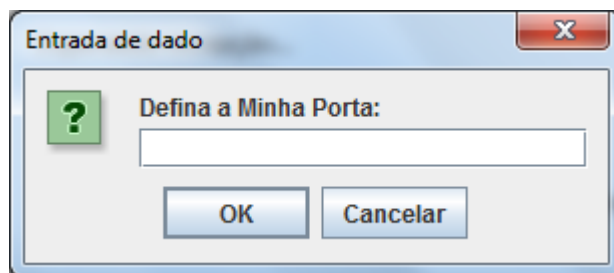


Figura 51 – Entrada de Dado: Defina a Minha Porta.

No momento em que o usuário escolhe a opção Enviar Artefato, é aberta a seguinte caixa de pesquisa, como mostra a Figura 52. Para que o artefato seja enviado, é necessário selecionar o diretório onde o arquivo com extensão PDF esteja localizado. Depois de selecionado o arquivo, basta escolher a opção abrir para que seja iniciado o envio.

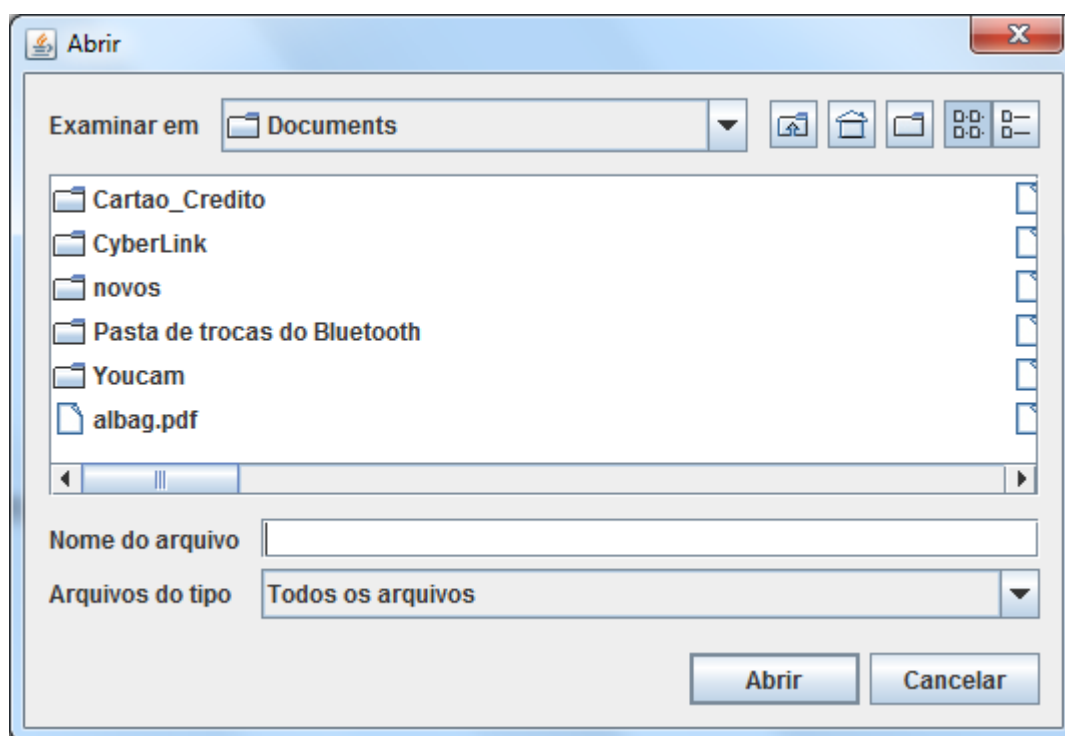


Figura 52 – Entrada de Dado: Defina a Minha Porta.

A opção Sair, do menu Arquivo, fecha o programa.

A Figura 53 apresenta as opções do menu Identificação do Gestor, a serem inseridas pelo usuário: Porta e Endereço IP.

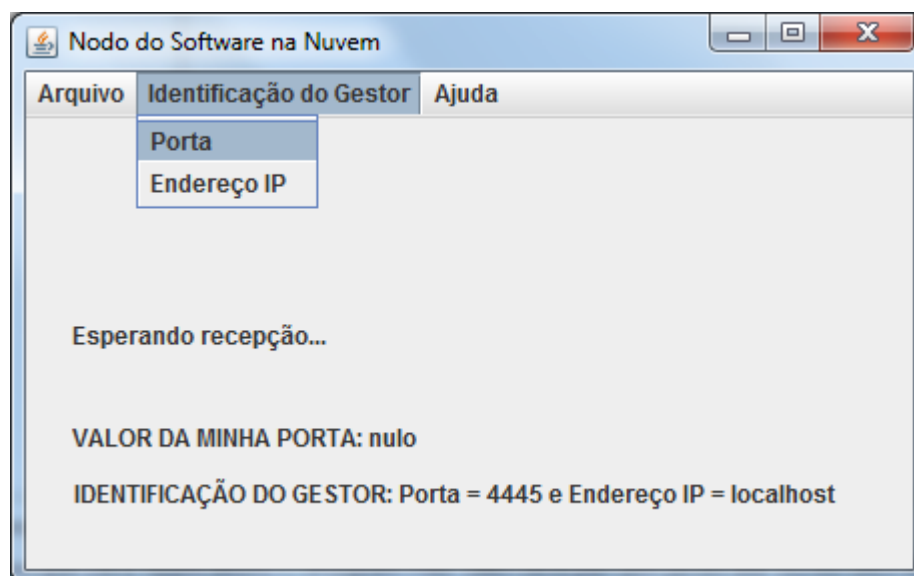


Figura 53 – Opções do menu Identificação do Gestor.

O usuário deverá inserir a porta utilizada pelo gestor, conforme a Figura 54 mostrada abaixo.

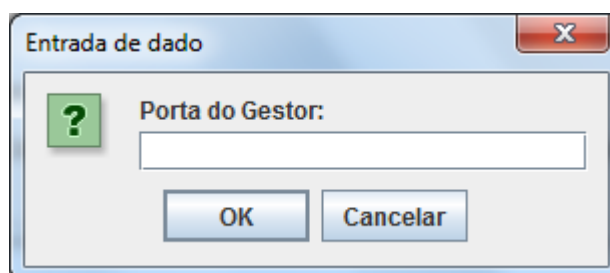


Figura 54 – Entrada de Dado: Defina a Minha Porta.

Após definida a porta do Gestor, o usuário deverá inserir o endereço IP do Gestor, conforme mostrado na Figura 55.

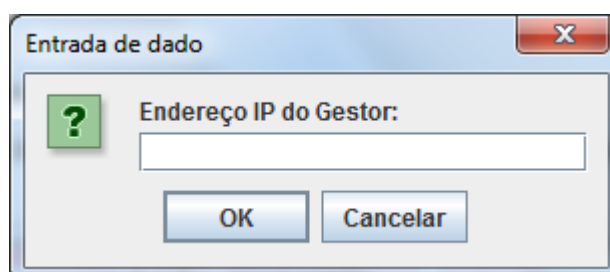


Figura 55 – Entrada de Dado: Endereço IP do Gestor.

O menu Ajuda apresenta a opção Sobre, conforme mostra a Figura 56.

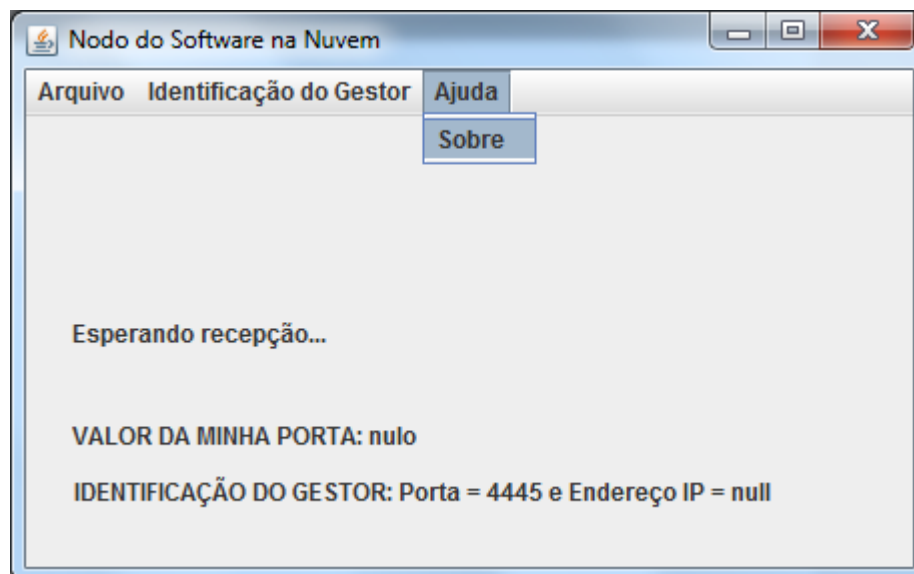


Figura 56 – Menu Ajuda: Sobre.

Na opção Sobre, é possível visualizar a versão do *software*, conforme ilustra a Figura 57.

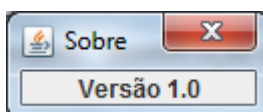


Figura 57 – Sobre: Versão 1.0.

4.9 Comentários

Neste capítulo foram expostos os detalhes do projeto físico das bases de dados gestsoftnuvem e lnebulosa. Também foram descritos, de forma sucinta, todos os métodos dos programas Gestor e Nodo de *Software* na Nuvem, bem como o algoritmo do modelo de inferência nebuloso implementado para executar o SIR e o SIS. Para finalizar este capítulo, foram apresentadas as telas da interface dos programas implementados.

No próximo capítulo, serão expostos os resultados dos testes realizados para ambos os sistemas de inferência. O escopo dos testes é detalhado em relação à presença do Gestor e apenas um Nodo e também com o Gestor e quatro Nodos.

5 TESTES E RESULTADOS

5.1 Introdução

Neste capítulo serão abordados aspectos da execução dos testes, bem como o desempenho apresentado por eles. Na segunda seção, os testes com o SIR são descritos detalhadamente. Já na terceira seção, os testes realizados com o SIS são relatados. O ambiente de testes é apresentado na quarta seção. A quinta seção exhibe o teste com o Gestor e apenas Um Nodo de *Software* na Nuvem, destacando o funcionamento do protocolo de rede. E, finalmente, a sexta seção descreve o teste realizado com o gestor de *software* na nuvem e os quatro Nodos.

5.2 Testes com o Sistema de Inferência Nebuloso de Rede

No sentido de se constatar a correção da inferência processada pelos sistemas nebulosos de rede e de segurança, tais sistemas também foram criados no ambiente MatLab para que se pudesse comparar o seu resultado com a inferência executada pelo Gestor codificada em Java.

A Tabela 28 apresenta os resultados da inferência para o sistema nebuloso de rede processados no MatLab. Nessa tabela, observam-se os valores de entrada iguais aos que foram executados pelo Gestor para efeitos de comparação. Uma bateria de dez rodadas de simulações foi executada e calculada a média aritmética, conforme mostrada na Tabela 28. Foram destacadas as três possíveis classificações para o IPR: 16,7%, índice que representa que não há problemas de rede, 50%, índice que representa que é provável que haja problemas de rede e 83,7%, índice que representa que a rede está com problemas.

Tabela 28 - Resultados do SIR executados pelo MatLab.

HOST	ATRASSO	SERVIÇOS	TENTATIVAS	IPR (%)
192.168.56.102	0,02747176415504	4	10	16,7
192.168.56.102	4,2788176029433	4	10	33,18
192.168.56.102	6,898388229101	4	10	44,68
192.168.56.102	8,8989899999777	4	10	50
192.168.56.102	9,234667840112	4	10	73,82
192.168.56.102	10	3	10	83,7

Para a inferência da rede, foram efetuados dois testes. O primeiro com os quatro nodos ligados, onde não foi diagnosticado nenhum problema de rede, conforme o resultado de 16,7% mostra. Outro teste foi executado e foi inferido que é provável que estivesse ocorrendo um problema de rede, conforme a resposta de 50% mostra. Para o segundo teste, foi desligada uma das máquinas e encontrou-se o índice de 83,7%, indicando que há problema de rede.

Comprovou-se que os resultados obtidos pelo Gestor foram bastante satisfatórios, pois eles foram praticamente iguais aos gerados pelo MatLab, conforme a tabela extraída pelo HeidiSQL indica. Destaca-se, por exemplo, o monitoramento de código 265 que apresenta o resultado do IPR igual a 16,666666666666668% calculado pelo Gestor e, observando-se o mesmo processamento (com as mesmas entradas), feitos pelo MatLab constata-se que o valor do IPR calculado foi de 16,7%, ou seja, resultado idêntico ao do Gestor se for considerado uma precisão de uma casa decimal.

Tabela 29 - Resultados obtidos pelo Gestor.

🔑 código	host	atraso	servicos	tentativas	indice_prob_rede
265	192.168.56.102	0,02747176415504346	4	10	16,666666666666668
270	192.168.56.102	10	4	10	49,99999999999997
304	192.168.56.102	10	3	10	83,33333333333334

5.3 Testes com o Sistema de Inferência Nebuloso de Segurança

Para a execução dos testes de segurança, foram alterados os parâmetros de entrada, uma vez que uma das entradas do SIS é a própria saída do SIR, o IPR.

A Tabela 30 a seguir mostra os resultados obtidos para outras entradas, onde IPS é o resultado obtido com a simulação feita no MatLab. Uma bateria de dez rodadas de simulações foi executada e calculada a média aritmética, conforme mostrada na Tabela 30. Foram destacadas as três possíveis classificações para o IPS: 16,3%, índice que representa que não há problemas de segurança, 50%, índice que representa que é provável que haja problemas de segurança; e 83,34%, índice que representa que o sistema apresenta problemas de segurança.

Para esta inferência de segurança, o primeiro teste executado foi apenas a simulação normal, onde não houve nenhum problema, tanto de rede (16,7%) nem de segurança (16,3%). Uma segunda bateria de testes foi executada, de modo que foi inferido que havia probabilidade de existir problema de segurança (50%) e, por fim, efetivamente, foi

comprovado que havia problema de segurança, com a inferência de 83,34%, quando foi enviado um artefato defeituoso para o Gestor.

Tabela 30 - Resultados do SIS executados pelo MatLab.

IPR	ACE	AFP	IPS (%)
16,7	0	0	16,3
33,18	0	0	34,20
50	100	0	50
73,82	0	100	74,14
50	100	100	83,34

Comprovou-se que os resultados obtidos pelo MatLab foram bastante satisfatórios, pois assemelharam-se com os gerados pelo software, conforme a tabela extraída pelo HeidiSQL indica.

Destaca-se, por exemplo, o monitoramento que apresenta o resultado do IPS igual a 16,666666666666666% calculado pelo Gestor e, observando-se o mesmo processamento (com as mesmas entradas), feitos pelo MatLab constata-se que o valor do IPS calculado foi de 16,3%, ou seja, resultado praticamente idêntico ao do Gestor se for considerado uma precisão de uma casa decimal.

Tabela 31- Resultados obtidos pelo Gestor.

host	atraso	servicos	tentativas	indice_prob_rede	artefatos_errados	artefatos_atrasados	indice_prob_seg
192.168.56.102	0,06907022753677806	0	1	16,666666666666668	0	0	16,666666666666666
192.168.56.103	10	1	10	50,000000000000002	0	0	50,000000000000002
192.168.56.102	10	2	10	50,000000000000002	100	100	83,33333333333334

Quando o sistema infere que há um problema de segurança, o monitoramento apresenta uma mensagem indicando que o host monitorado está com problema de segurança. O monitoramento é abortado, conforme ilustra a Figura 58.

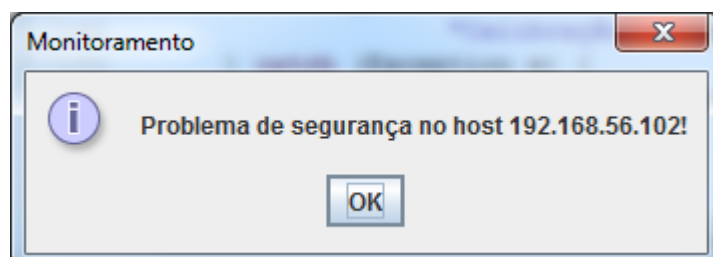


Figura 58 - Problema de Segurança Detectado.

5.4 Ambiente de Teste da Computação em Nuvem

Para esta dissertação foi preparado um ambiente contendo quatro máquinas virtuais configuradas pelo Oracle VM VirtualBox e uma máquina real. As máquinas virtuais representam os Nodos. A máquina real representa o Gestor. Para a simulação e execução dos testes foram preparadas três máquinas virtuais (Nodo1, Nodo2 e Nodo3) rodando o Sistema Operacional Windows XP, Service Pack 3. A outra máquina virtual, que completa a composição dos Nodos, foi configurada com o sistema operacional Linux, com a distribuição Ubuntu (Nodo4).

Para o Gestor (Gestor), foi utilizada uma máquina real, com o Sistema Operacional Windows 7 64 Bits. A atribuição de IPs é apresentada na Tabela 32:

Tabela 32 - Alocação de IPs para os Nodos e Gestor.

IP	HOST
192.168.56.101	Gestor
192.168.56.102	Nodo1
192.168.56.103	Nodo2
192.168.56.104	Nodo3
192.168.56.105	Nodo4

5.5 Testes com o Gestor e um Nodo de Software na Nuvem

O escopo de testes com o Gestor e um Nodo foi realizado contemplando as cinco etapas, descritas a seguir.

A primeira etapa consiste em executar a calibração do Nodo processada pelo Gestor, comparando e apresentando os valores dos conteúdos referentes à tabela host (trecho da interface do aplicativo HeidiSQL) da máquina virtual Nodo 3, antes e após a calibração, conforme indicado na sequência das Figuras 58 e 59.

endereço_ip	servicos	atraso
192.168.56.102	0	-1

Figura 59 - Conteúdo da Tabela Host antes da calibração.

endereço_ip	servicos	atraso
192.168.56.102	0	64,705111

Figura 60 - Conteúdo da Tabela Host após a calibração.

Foi inserido um atraso com um valor default de -1 para indicar que a calibração ainda não foi feita e, portanto, não existe valor válido de atraso para o host considerado. Após a calibração, o Gestor atualiza o Banco de Dados ao colocar um valor válido de atraso medido no registro correspondente ao *host* calibrado. Analogamente, o campo *servicos*, inicialmente possui o valor *default* 0, o qual é alterado para a quantidade de serviços ativos que o *host* possui após a sua calibração.

A segunda etapa do teste consiste em um Nodo enviar um dado artefato para o Gestor. Nas Figuras 61 e 62, são exibidas as telas de evolução do envio de um arquivo em formato PDF, o qual se trata de um artefato. Pode-se constatar na Figura 62, que o arquivo chegou íntegro e sem problemas.

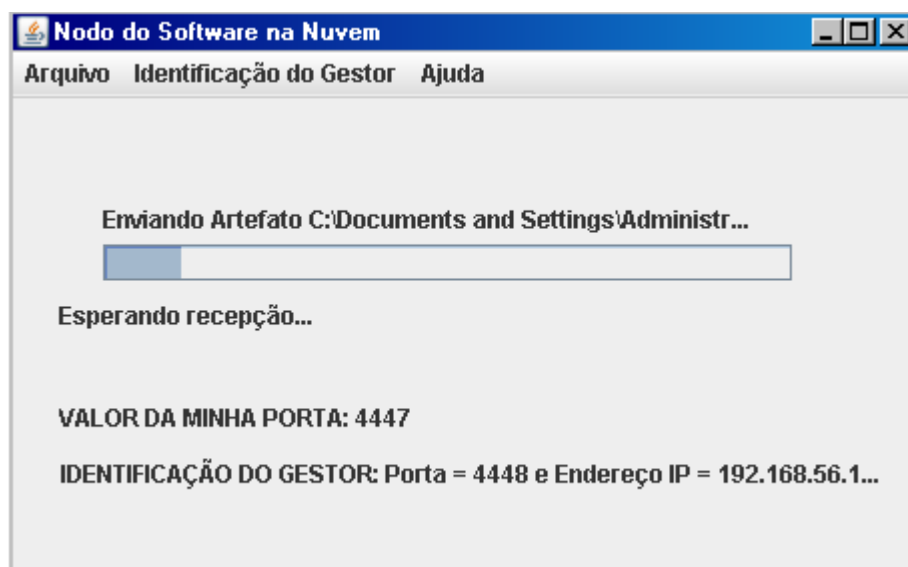


Figura 61- Início do Nodo enviando Artefato.

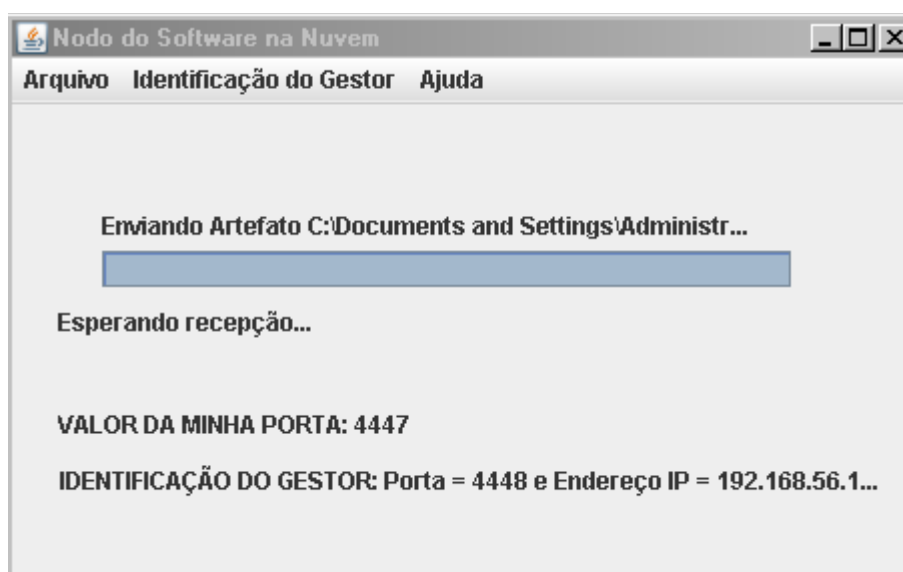


Figura 62 - Término do envio do Artefato.

Após o término do envio do Artefato pelo Nodo, a janela da Figura 63 é apresentada pelo programa Gestor, indicando que um arquivo de artefato foi recebido corretamente. Tal janela exibe ainda o endereço IP do Nodo que enviou o artefato em questão.

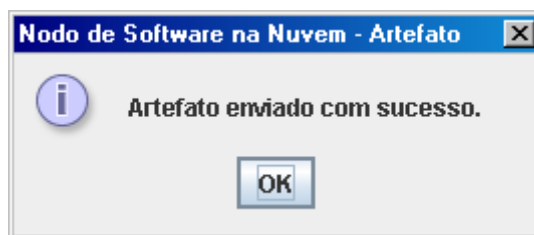


Figura 63 - Artefato enviado.

Do lado do Gestor, a mensagem exibida ao término do recebimento do Artefato é mostrada na Figura 64.

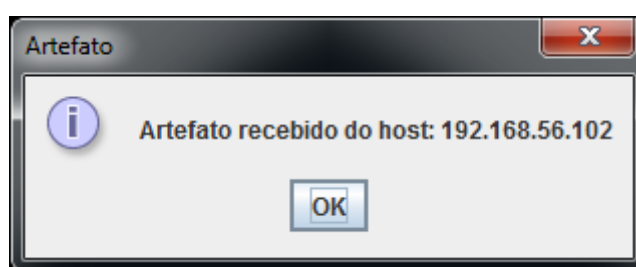


Figura 64 - Artefato enviado pelo Nodo recebido com êxito pelo Gestor.

Para ilustrar o recebimento correto do Artefato, é mostrado o diretório C:\temp no lado do Gestor, onde foi designado para o recebimento do mesmo, conforme indicado na Figura 65. Foi necessário criar o diretório temp no diretório C do Gestor, para que o programa pudesse receber corretamente os artefatos. A formatação do nome do arquivo é composta pelos seguintes elementos:

Artefato = para indicar que o arquivo recebido é um artefato.

Data = a função `dataHora.getTime()` utilizada no Java converte a data e o horário para milissegundos.

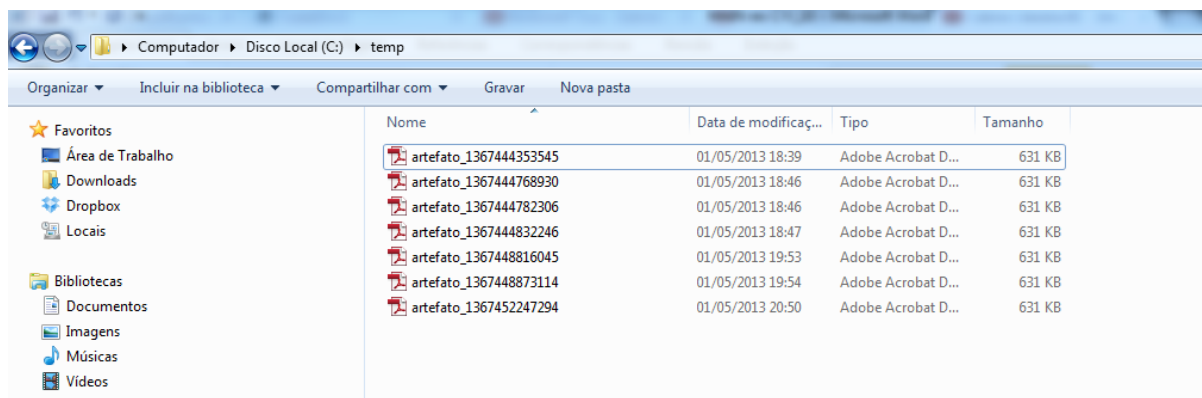


Figura 65- Diretório do Gestor contendo os Artefatos enviados pelo Nodo.

Na terceira etapa do teste são apresentadas duas capturas das telas referentes às quatro operações básicas de acesso a qualquer banco de dados (inserção, atualização, exclusão e consulta), mostradas nas Figuras 66 e 67. O acesso ao banco de dados é processado pelo programa Gestor apenas. O Gestor manipula sete tabelas cujas informações são usadas para controlar a atividade de gestão de desenvolvimento do *software*, a saber: Sistema, Equipe, Fase, Atividade, Host, Serviço e Monitoramento. No sentido de exemplificar tal funcionalidade de manipulação dessas tabelas, foram destacadas nas figuras a seguir apenas a gerência da tabela Equipe.



Figura 66 - Manter Atividade

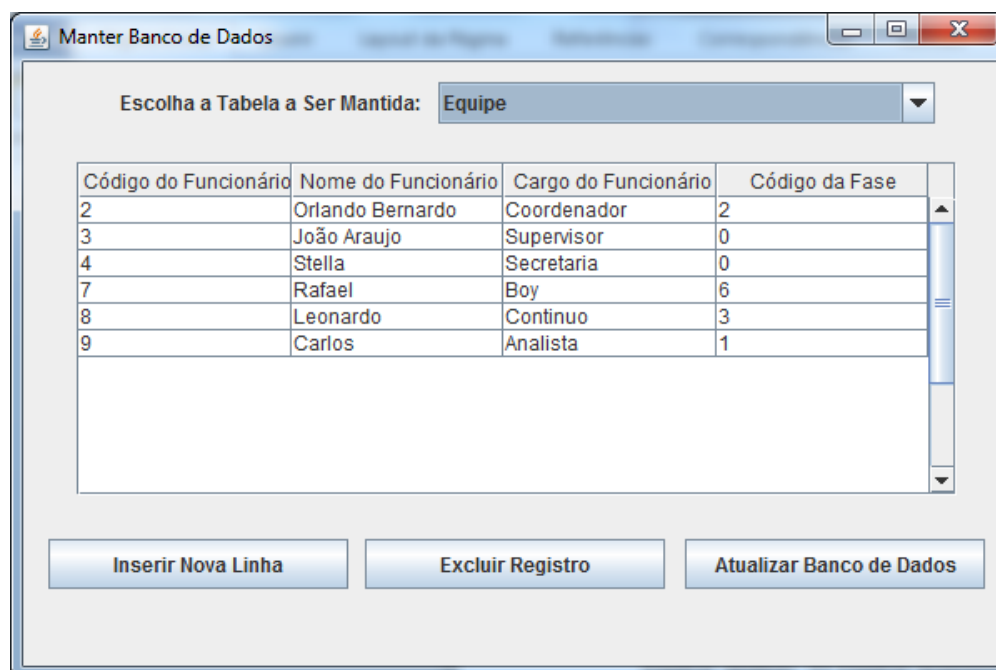


Figura 67 - Manter Equipe.

Na quarta etapa do teste, apresenta-se a definição dos parâmetros de endereço IP e porta para o Gestor executada pelo programa Nodo. Podem-se visualizar nas Figuras 68 e 69 os referidos parâmetros que foram alterados.

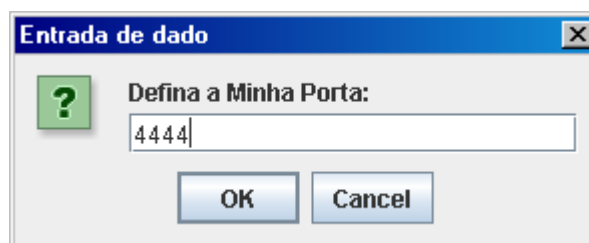


Figura 68 - Definindo Porta do Gestor.

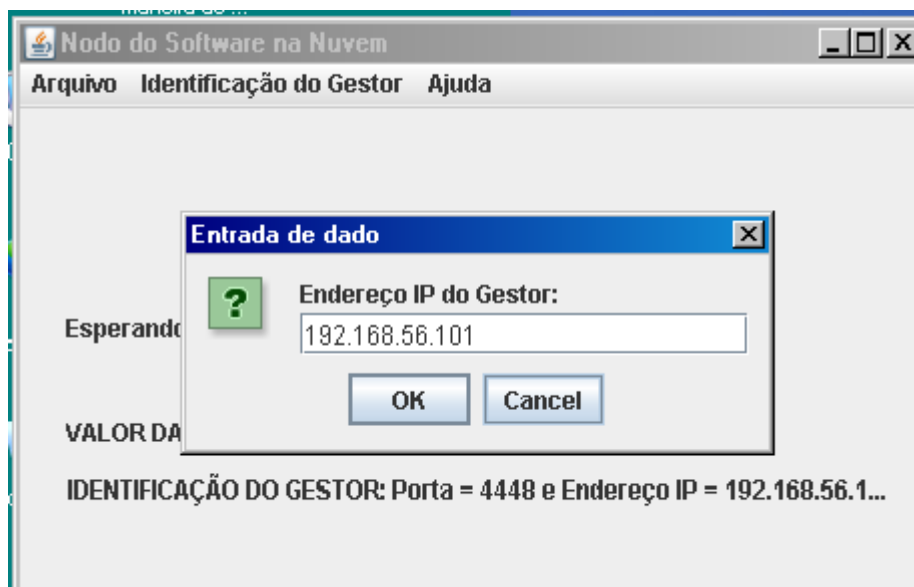


Figura 69 - Definindo IP do Gestor.

Por fim, a última parte do teste, referente à monitoração feita pelo Gestor, é apresentada a tela de execução do algoritmo e, posteriormente, é exibido o conteúdo da tabela Monitoramento, com destaque nos valores calculados pelo SIR e SIS conforme mostrado na Figura 70 e Tabela 33.

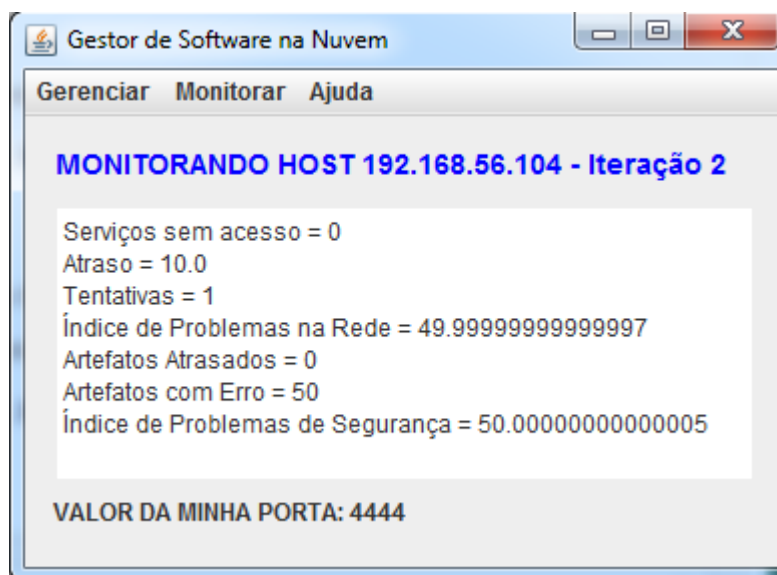


Figura 70 – Iteração 2 do Host.

Tabela 33 – Tabela Monitoramento no HeidiSQL.

🔑	codigo	host	atraso	servicos	tentativas	indice_prob_rede	artefatos_errados	artefatos_atrasados	indice_prob_seg
	388	192.168.56.104	10	0	1	49,99999999999997	50	0	50,00000000000005
	392	192.168.56.104	10	0	1	49,99999999999997	50	0	50,00000000000005

É possível observar, que o IPR e IPS são mostrados na tela do *software* e atualizados nos registros do banco de dados. Na tela do Gestor de *Software* na Nuvem é apresentada a monitoração do host em questão (cujo IP é 192.168.56.104), e os valores das entradas medidas para o SIS e para o SIR. O cancelamento da monitoração aguarda a finalização do cálculo da iteração corrente e cancela a próxima iteração.

5.6 Testes com o Gestor e Vários Nodos de *Software* na Nuvem

O escopo desta simulação aborda a configuração de testes realizados com o Gestor, sendo executado na máquina real e quatro Nodos, os quais são executados em máquinas virtuais.

A primeira etapa dos testes consiste em testar a Calibração. A Figura 71 apresenta a tabela Host antes da Calibração, com as colunas: Endereço_IP, onde são listados os endereços IP das máquinas que rodam o programa Nodo; a coluna Serviços, contendo os valores “0”, que indicam que o sistema ainda não foi iniciado. Por fim, a coluna Atraso contendo os valores “-1” indica que ainda não há atraso registrado.

🔑	endereço_ip	servicos	atraso
	192.168.56.102	0	-1
	192.168.56.103	0	-1
	192.168.56.104	0	-1
	192.168.56.105	0	-1

Figura 71 - Tabela Host antes da Calibração

A tabela Serviços, mostrada na Figura 72, apresenta as colunas: Código, contendo o código de cada registro; Status, contendo o estado do registro, Host, comportando o endereço IP dos Nodos e Porta, contendo o número da Porta onde um dado serviço é executado.

🔑	codigo	status	host	porta
	2	Fase	192.168.56.102	4445
	3	Ativo	192.168.56.102	8080
	4	Ativo	192.168.56.102	4447
	5	Ativo	192.168.56.102	4448
	7	Ativo	192.168.56.103	3306
	8	Fase	192.168.56.103	4445
	9	Ativo	192.168.56.102	4450
	10	Fase	192.168.56.104	4445
	11	Ativo	192.168.56.105	7575
	12	Ativo	192.168.56.104	5132
	13	Fase	192.168.56.105	4445
	14	Ativo	192.168.56.105	2013
	15	Ativo	192.168.56.105	8888

Figura 72: Tabela Serviços

A Figura 73 apresenta os resultados da Tabela Host após a Calibração. São mostrados os conteúdos calculados e cadastrados automaticamente no banco de dados. Para ilustrar, é possível verificar que o *host* 192.168.56.102 possui 4 serviços, e o maior atraso medido, sendo de quase 6 ms.

🔑	endereco_ip	servicos	atraso
	192.168.56.102	4	5,81118
	192.168.56.103	1	3,832621
	192.168.56.104	1	2,039245
	192.168.56.105	2	3,153914

Figura 73: Tabela Host após Calibração

Tabela 34 - Monitoramento do HeidiSQL

🔑	codigo	host	atraso	servicos	tentativas	indice_prob_rede	artefatos_errados	artefatos_atrasados	indice_prob_seg
	399	192.168.56.102	0,3135175287358158	0	1	16,666666666666668	0	0	16,666666666666666
	400	192.168.56.103	10	0	1	49,999999999999997	50	0	50,000000000000005
	401	192.168.56.104	10	1	10	50,000000000000002	0	0	50,000000000000002
	402	192.168.56.105	10	2	10	50,000000000000002	0	0	50,000000000000002
	403	192.168.56.102	0,048961430840379815	0	1	16,666666666666668	0	0	16,666666666666666
	404	192.168.56.103	10	0	1	49,999999999999997	50	0	50,000000000000005

A Tabela 34 mostra o monitoramento do Gestor exibido através da interface do HeidiSQL. Nesta tabela, são apresentados os resultados de todos os quatro nodos monitorados

(conforme a coluna host indica) e os valores das entradas medidas para executar o SIR e o SIS, além dos valores de IPR e IPS inferidos. Neste exemplo, a monitoração foi cancelada após a segunda iteração para o Nodo cujo IP é o 192.168.56.103.

Por suas características, *Man In The Middle*, *Sniffing* e o *Mapping*, são tipos de ataque que ocorrem fora do domínio do monitoramento proposto neste trabalho e portanto, foram excluídos do escopo. A Tabela 35 apresenta os ataques simulados nesta dissertação.

Tabela 35 - Ataques Simulados.

	1. Spoofing.	2. Trojans	3. DoS e DDoS.
Atraso			x
Quantidade de Serviços sem Acesso		x	x
Número de Tentativas de Conexão		x	x
Artefatos Com Erro	x	x	x
Artefatos Fora do Prazo			

Spoofing: Por emular um endereço que não pode ser alcançado pelo segmento de rede de onde este ataque está sendo gerado, como resultados serão gerados artefatos com erros.

Trojans: *Trojans* já instalados ou mapear possíveis alvos podem dificultar o acesso a serviços, aumentar significativamente o número de tentativas de conexão e gerar artefato com erros.

DoS e DDoS: Ataques de negação de serviço tem por natureza causar atrasos nos pacotes em trânsito na rede, fazem com que serviços fiquem inacessíveis e produzem artefatos com erro.

Para simular o *spoofing*, foi alterado o IP do nodo. Com isso, o IPR detectou que a máquina destino não estava acessível, e o sistema acusou problema de rede. Para simular os *Trojans*, foi desligada uma das máquinas e assumido que o arquivo não chegou de forma correta e atrasado. Para simular o DoS e o DDoS, foi desligada uma das máquinas, e o IPR detectou problemas de rede. Os resultados foram expostos na Tabela 36 abaixo:

Tabela 36 - Resultados.

Ataque	Máquina	IPR	IPS
<i>Spoofing</i>	1	16%	50%
<i>Trojans</i>	1	16%	83%
DoS	1	83%	50%

5.7 Comentários

Neste capítulo foram descritas as etapas dos testes, onde foi idealizado um experimento com quatro máquinas virtuais criadas para implantar os sistemas de inferência nebulosos (de rede e de segurança). As máquinas virtuais conectadas em rede são os nós (ou Nodos) da nuvem do estudo de caso implementado, ou seja, do Gestor de *Software* na Nuvem (*gestsoftnuvem*). No sentido de observar o comportamento dos sistemas de inferência nebulosos para constatar a sua eficácia, foram introduzidos erros propositadamente em uma das máquinas virtuais para causar um mau funcionamento na nuvem. O primeiro teste foi simplesmente desligar uma das máquinas virtuais, provocando uma detecção de possível problema na rede, visto que essa máquina desligada ficaria totalmente sem acesso. Já o segundo teste seria manter as máquinas em funcionamento, porém programando o cliente do *gestsoftnuvem* de uma delas para enviar um artefato defeituoso para o gestor. As quatro máquinas virtuais foram instaladas com o Windows XP. A máquina Gestor é uma máquina real, rodando instalada com Windows 7, 64 bits. Por fim, pode-se dizer que os testes apresentaram resultados satisfatórios, que comprovaram a viabilidade do emprego da técnica de inferência desenvolvida.

No próximo e último capítulo deste estudo, são relacionadas as conclusões e apresentadas ideias para possíveis trabalhos futuros.

6 CONCLUSÕES

A facilidade e demanda de conectividade de dispositivos em uma rede de computadores eleva a necessidade de segurança em relação ao armazenamento e disponibilidade de informações. A preocupação com o compartilhamento de informações de forma íntegra tem papel fundamental no atual cenário da evolução tecnológica.

Neste estudo de caso foi implementado um sistema nebuloso de detecção de intrusos para computação em nuvem de modo a gerir o desenvolvimento de um *software* de modo distribuído. Para tal, foi considerado que desenvolvedores interagem através da nuvem para o desenvolvimento de um produto de *software*, de forma colaborativa. Cada um desses desenvolvedores opera uma máquina, que executada o programa Nodo e enviam os artefatos para o gerente, que opera a máquina que executa o programa Gestor.

As simulações foram executadas em quatro máquinas virtuais rodando Windows XP, com o programa Nodo. O Gestor foi executado em uma máquina real, com sistema operacional Windows 7.

A implementação do banco de dados foi feita utilizando o aplicativo de interface gráfica HeidiSQL (HEIDISQL, 2012) para o sistema gerenciador de banco de dados distribuído MySQL (MYSQL, 2012). O código, escrito em Java, atualiza e preenche os registros, de acordo com a execução das iterações de monitoramento das máquinas ou por iniciativa do próprio usuário do gestsoftnuvem.

Os resultados obtidos por meio dos Sistemas de Inferência de Rede (SIR) e de Segurança (SIS), de modo geral, foram bastante satisfatórios e foi possível diagnosticar se os problemas apresentados eram de rede ou de segurança, uma vez que a partir das regras e variáveis linguísticas pré-definidas, possibilitaram inferir se o sistema estava sem problemas, com prováveis problemas e com problemas.

Melhorias são sempre bem vindas em qualquer projeto e processo. Estudos futuros podem melhorar ou estender a pesquisa realizada. Logo a seguir são listadas algumas sugestões e possibilidades para trabalhos futuros:

- (1) Criar um sistema de verificação da integridade do artefato transmitido, substituindo a necessidade do artefato ser verificado manualmente, através de inferência humana. Uma sugestão é criar uma técnica para detecção de erros

semelhante à utilizada em protocolos, de modo a assegurar-se que o arquivo do artefato não foi recebido corrompido.

(2) Desenvolver uma rotina de recebimento de notificações de problemas, por e-mail, de modo que seja possível para o gerente ou para outro responsável pelo projeto ter ciência das falhas de qualquer lugar, mesmo fora da nuvem e para isso, poder-se-ia utilizar a API JavaMail para enviar mensagens de correio eletrônico ao administrador do sistema.

(3) Autenticação segura na máquina do Gestor e Nodos. Objetivando aumentar o nível de segurança do sistema, criar um mecanismo de autenticação utilizando *tokens*, isto é, dispositivos físicos (o formato assemelha-se a chaves e *pendrives*) que geram uma combinação de números temporária para autenticação segura do usuário. Essa senha é válida apenas por um determinado tempo, dificultando a ação de invasores.

(4) Implementação de um módulo gerador de Relatórios Gerenciais utilizando o conteúdo do banco de dados. É desejável desenvolver uma forma de criar relatórios dinâmicos, que sejam personalizáveis, contendo os dados de acordo com os atributos que os usuários desejem visualizar. Esse módulo gerador de relatórios dinâmicos deve ser de fácil manipulação, e facilite o trabalho dos usuários, contendo os dados mais frequentes utilizados por eles, tornando assim o trabalho colaborativo mais eficiente. É esperado que a personalização de relatórios, venha a resultar em melhores níveis de produtividade e satisfação dos usuários.

(5) Implementar um módulo de Sistema de Prevenção de Intrusos (SPI ou IPS, do inglês, *Intrusion Prevention System*), de modo que trabalhe em conjunto com o SDI já implementado, garantindo a potencialização da segurança do projeto. O SPI ideal é aquele que consegue detectar o tráfego malicioso em toda a rede e bloqueá-lo logo na sua origem, impedindo que infecte o sistema.

(6) Como última melhoria, manter a compatibilidade automática entre as informações das tabelas do banco de dados. Por exemplo, ao se cadastrar um novo membro da tabela Equipe, esse deve ser associado a uma fase do *software* em

desenvolvimento, logo, tal fase, obrigatoriamente, já deve estar atualizada e constar na tabela Fase.

REFERÊNCIAS

- ABNT 2001 – Associação Brasileira de Normas Técnicas, ABNT ISO/IEC 17799 – Tecnologia da Informação – Código de prática para gestão da segurança da informação, Agosto 2001.
- BARBOSA, A. S.; MORAES, L. F. M. de. *Sistemas de Detecção de Intrusão - Seminários Ravel -CPS760*. 2000.
- BENSO, A. C. Gerenciamento de redes móveis. Tese (Doutorado em Ciência da Computação), Universidade Federal do Rio Grande do Sul, 2003.
- BLACK, T. L. *Comparação de Ferramentas de Gerenciamento de Redes*. Dissertação (Mestrado) Universidade Federal do Rio Grande do Sul, 2008.
- BOUML. BOUML. *Disponível em: <http://www.bouml.fr/>. 2012.*
- BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. *Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities*. 2009.
- CARDANA, J. M. A. *Analisador Comportamental de Rede*. Dissertação (Mestrado) Universidade de Lisboa, 2006.
- CIURANA., E. *Developing with Google App Engine*. Apress, Berkely, CA, USA. 2009.
- DENNING, D. *An Intrusion Detection System*. 1986.
- DETSCH, A. Uma Arquitetura para Incorporação Modular de Aspectos de Segurança em Aplicações Peer-to-Peer. Universidade do Vale do Rio dos Sinos, São Leopoldo. Dissertação de Mestrado, 2005.
- DUMONT, C. E. S., *Segurança Computacional: Segurança em Servidores Linux em Camadas*. Universidade Federal de Lavras. 2006.
- FELICIANO, G. et al. *Uma Arquitetura para Gerência de Identidades em Nuvens Híbridas*. 2011.
- FERREIRA, F. *Segurança da Informação*. Rio de Janeiro: Ciência Moderna, 2003.
- FONSECA, O. L. H. Aplicação de Métodos de Análise Espacial e da Teoria dos Conjuntos Nebulosos em Estudo Sobre Pobreza. 146p. Dissertação (Mestrado em Engenharia de Computação). Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2003.

- FOSTER, I. *What is the Grid? A Three Point Checklist*. 2002.
- HATCH, B.; LEE, J.; KURTZ, G. *Hackers Linux Expostos: Segredos e Soluções para a Segurança do Linux*. São Paulo: Makron Books, 2002.
- HEIDISQL. HEIDISQL. *Disponível em: <http://www.heidisql.com/>*. 2012.
- IDRIS, N. B.; SHANMUGAM, B. Artificial Intelligence Techniques Applied to Intrusion Detection. In: Annual IEE Indicon, 2005. Chennai. Proceedings. IEEE, 2005. p. 52-55.
- INFO, 2012. Disponível em <http://info.abril.com.br/noticias/internet/usuarios-relatam-falhas-em-servicos-do-google-no-brasil-26112012-34.shl>. 2012
- JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999.
- JAMSA, D. K. *Cloud Computing*. [S.l.]: Jones and Bartlett Learning, 2012.
- JI, C. Y.; ALMEIDA, N. N. de; BERNARDO FILHO, O. Lógica nebulosa aplicada a um sistema de detecção de intrusos para computação em nuvem. II CBSF - *Segundo Congresso Brasileiro de Sistemas Fuzzy*, v. 1, n. 1, p. 1–18, Novembro 2012.
- KABIRI, P.; GHORBANI, A. A. Research on Intrusion Detection and Response: A Survey. *International Journal of Network Security*, p. 84-102. Junho, 2005.
- LARSEN, P. M. *Industrial Applications of Fuzzy Logic Control*. London: Academic Press, 1981.
- Lawrence Livermore National Laboratory. *Network Intrusion Detector (NID) Overview*. Disponível em <http://ciac.llnl.gov/cstc/nid>. 2012.
- MANDANI, E. H. Application of Fuzzy Algorithms for Control of Simple Dynamic Plant. *Proceedings of the IEE Control and Science*, v. 121, p. 298-316, 1974.
- MATHWORKS. MATHWORKS. *Disponível em: <http://www.mathworks.com/help/toolbox/fuzzy/fp351dup8.html>*. 2012.
- MATLAB. *Matlab*. Disponível em: <http://www.mathworks.com/products/matlab/>. 2012.
- MCCLURE, S.; SCAMBRAY, J.; KURTZ, G. *Hackers expostos: Segredos e Soluções para a Segurança de Redes*. São Paulo: Makron Books, 2000.
- MELL, P. ; GRANCE, T. *The NIST definition of cloud computing*. 2009.
- National Institute of Standards and Technology.

- MENDEL, J. M. *Fuzzy Logic Systems for Engineering: A Tutorial*. 1995.
- MILLER, P.; INOUE, A. *Collaborative intrusion detection system*. 2003.
- MYSQL. MySQL. Disponível em <http://www.mysql.com/>. 2013
- NAKAMURA, E. T., GEUS, P.L. *Segurança de redes: em ambientes cooperativos*. 2. ed. São Paulo: Futura, 2003.
- NEO. Núcleo de Estudos em Otimização. Disponível em <http://www.lps.usp.br/neo/fuzzy.2008>.
- NetRanger 2.2.1. *Cisco Systems, Inc. NetRanger Intrusion Detection System Technical Overview*. Disponível em: <http://www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr/index.htm>. 2012.
- NID. *Lawrence Livermore National Laboratory*. 2012. Disponível em: http://en.wikipedia.org/wiki/Lawrence_Livermore_National_Laboratory
- OLIVEIRA JR., H. A. *Lógica Difusa: Aspectos Práticos e Aplicações*. Rio de Janeiro. : Ed. Interciência, 1999. 192p.
- OLIVEIRA, C. M., COSTA, M. R. *Rede Local, Utilizando a RS-232*. 1995. Projeto de Graduação do Curso de Engenharia de Sistemas e Computação - Universidade do Estado do Rio de Janeiro.
- OPENAM. Open Source Access Management. Disponível em: https://access.redhat.com/site/documentation/enUS/Red_Hat_JBoss_Portal/6.0/html/Reference_Guide/sect-Reference_Guide-SSO_Single_Sign_On_-OpenAM.html. 2012.
- PAXSON, V. Bro: *A System for Detecting Network Intruders in Real-Time*. 1998.
- PRESSMAN, Roger S. *Engenharia de software*. 6ª ed. Porto Alegre: Bookman, 2006.
- PTACEK, T. H.; NEWSHAM, T. N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. 1998.
- RUSCHEL, H.; ZANOTTO, M. S.; MOTA, W. C. da. *Computação em Nuvem*. 2010.
- SALESFORCE. Salesforce. Disponível em <http://www.salesforce.com/>. 2010.
- SAYDAM; MAGEDANZ, T. *From Networks and Network Management into Service and Service Management*. *Journal of Networks and System Management*, v. 4, n. 4, p. 345–348, December 1996.

- SCHNEIER, B. *Segurança com Segredos e Mentiras Sobre a Proteção na Vida Digital*. Rio de Janeiro: Campus, 2000.
- SILVA, J. R. C. da. *Sistemas de Detecção de Intrusão com Técnicas de Inteligência Artificial*. 2011.
- SOMMERVILLE, I. *Software Engineering*. [S.l.]: Addison-Wesley, 2007.
- SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO., J. C. *Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios*. 2010.
- SOUZA, F. J. Apostila de Aula de Inteligência Artificial do Programa de Pós-Graduação em Engenharia de Computação da UERJ. Rio de Janeiro, 2001.
- TAH, J.H.M.; CARR, V. A proposal for construction project risk assessment using fuzzy logic. In : *Construction Management and Economics* 18, p491-500, 2000
- TANEMBAUM, A. S. *Redes de Computadores*. [S.l.]: Campus, 2003.
- TÁPIA, M. *Lógica Difusa*. Disponível em <http://www.inf.ufsc.br/mauro/ine5377/leituras/AptFuzzy.ppt>. 2008.
- TANSCHKEIT, R. *Fundamentos de Lógica Fuzzy e Controle Fuzzy*. Apostila de aula do Departamento de Engenharia Eletrônica da PUC-RJ, Rio de Janeiro, 1999.
- TILLAPART, P. THUMTHAWATWORM, T.; SANTIPRABHOB, P. Fuzzy Intrusion Detection System. *AU Journal of Techonology*. P 109-115, Outubro, 2002.
- UML. UML. Disponível em: <http://www.uml.org/>. 2012.
- VECCHIOLA, C.; CHU, X.; BUYYA., R. *Aneka: A Software Platform for .NET-based Cloud Computing*. W. Gentsch, L. Grandinetti, G. Joubert (Eds.). *High Speed and Large Scale Scientific Computing*. IOS Press, Amsterdam, Netherlands.
- VIGNA, G.; KEMMERER, R. A. *NetSTAT: A Network-based Intrusion Detection Approach*. 2007.
- VIRTI, E. S.; *Implementação de um IDS Utilizando SNMP e Lógica Difusa*. Universidade Federal do Rio Grande do Sul, 2007.
- VIRTUALBOX. VIRTUALBOX. Disponível em: <https://www.virtualbox.org/>. 2012.
- WALKER, ROBERT J.; HOLMES, R.; HEDGELAND, IAN; KAPUR, PUNEET; SMITH,

XIA, H. et al. *A Subjective Trust Management Model with Multiple Decision Factors for MANET based on AHP and Fuzzy Logic Rules*. 2011.

ZADEH, L. A. Fuzzy Sets. *Information and Control* 8, p. 338-353 (1965).

ZHOU, Q. et al. A Novel Approach to Manage Trust in Ad Hoc Networks. 2007. International Conference on Convergence Information Technology. University of Electronic Science and Technology of China, pp. 295–300, IEEE Computer Society (2007).

APÊNDICE A — FONTES DO GESTOR DE SOFTWARE NA NUVEM

A.1 PACOTE CONTROLE

Classe *Conexao.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável pela abertura de conexão com o banco de dados. */
*****/

package controle;

import java.sql.*;
import javax.swing.JOptionPane;

public class Conexao {

//Método estático que retorna nova conexão
public static Connection getConnection() throws SQLException {

try {
    Class.forName("com.mysql.jdbc.Driver");// Registrando o driver
    // Retorna a conexão
    return DriverManager.getConnection("jdbc:mysql://localhost:3306/gestsoftnuvem",
"root", "1234");
}
catch (ClassNotFoundException e) {
    JOptionPane.showMessageDialog(null, "Não foi possível se conectar ao
banco!\nInformações sobre o erro: "
+ e, "Conexao", JOptionPane.ERROR_MESSAGE);

    return null;
}
}
}

```

Classe *GestorSoft.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável pelo acionamento das funcionalidades do Gestor, */
/* promovendo a interação entre os pacotes de interface e o pacote modelo.. */
*****/

package controle;

```

```

import java.sql.*;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import interface_usuario.*;
import interface_rede.*;
import DAO.*;
import modelo.*;

public class GestorSoft {

InterfaceManterSistema ims = new InterfaceManterSistema();
InterfaceManterEquipe ime = new InterfaceManterEquipe();
InterfaceManterFase imf = new InterfaceManterFase();
InterfaceManterAtividade ima = new InterfaceManterAtividade();
InterfaceManterHost imh = new InterfaceManterHost();
InterfaceManterServico imserv = new InterfaceManterServico();
InterfaceManterMonitoramento imm = new InterfaceManterMonitoramento();
int MinhaPorta = -1;
Servidor serv;

// Método responsável por obter um novo valor de porta do usuário e defini-lo para
o Gestor
// Em seguida, uma nova instância de servidor é criada com este novo valor de porta
public void alterarPorta (JLabel jLabelPorta)
{
    String p = JOptionPane.showInputDialog(jLabelPorta, "Defina a Minha Porta:",
//Caixa de diálogo
        "Entrada de dado", JOptionPane.QUESTION_MESSAGE);
    if (!p.equals(null))
    {
        MinhaPorta = (Integer.parseInt(p));
        jLabelPorta.setText("VALOR DA MINHA PORTA: " + MinhaPorta);
        if (serv != null) serv.setListening(false);
        serv = new Servidor(MinhaPorta);
        serv.start();
    }
}

// Método responsável pela calibração dos nodos do software na nuvem.
// Tal método obtém os valores de atraso e serviços acessados de cada nodo.
public void calibrar()
{
    Cliente c = new Cliente();
    c.executaCalibracao();
}

// Método responsável por fazer o acesso à tabela Sistema do banco de dados
// e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarSistema (JTable jTable)
{
    Sistema[] s;

    try {
        SistemaDAO sistemaDAO = new SistemaDAO();

        s = sistemaDAO.retrieveAll();
        ims.preencheTabelaSistema(s, jTable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por fazer o acesso à tabela Equipe do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarEquipe (JTable jTable)
{

```



```

    Equipe[] e;

try {
    EquipeDAO equipeDAO = new EquipeDAO();

    e = equipeDAO.retrieveAll();
    ime.preencheTabelaEquipe(e, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por fazer o acesso à tabela Fase do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarFase (JTable jTable)
{
    Fase[] f;

try {
    FaseDAO faseDAO = new FaseDAO();

    f = faseDAO.retrieveAll();
    imf.preencheTabelaFase(f, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por fazer o acesso à tabela Atividade do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarAtividade (JTable jTable)
{
    Atividade[] a;

try {
    AtividadeDAO atividadeDAO = new AtividadeDAO();

    a = atividadeDAO.retrieveAll();
    ima.preencheTabelaAtividade(a, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por fazer o acesso à tabela Host do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarHost (JTable jTable) //Consulta Host da máquina
{
    Host[] h;

try {
    HostDAO hostDAO = new HostDAO();

    h = hostDAO.retrieveAll();
    imh.preencheTabelaHost(h, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por fazer o acesso à tabela Serviço do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarServico (JTable jTable)
{

```

```

        Servico[] serv;

try {
    ServicoDAO servicoDAO = new ServicoDAO();

    serv = servicoDAO.retrieveAll();
    imserv.preencheTabelaServico(serv, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por fazer o acesso à tabela Monitoramento do banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void consultarMonitoramento (JTable jTable)
{
    Monitoramento[] m;

try {
    MonitoramentoDAO monitoramentoDAO = new MonitoramentoDAO();

    m = monitoramentoDAO.retrieveAll();
    imm.preencheTabelaMonitoramento(m, jTable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por inserir uma linha em branco na interface da tabela
Sistema.
public void inserirLinhaSistema(JTable jTable)
{
    ims.criaLinhaTabelaSistema(jtable);
}

//Método responsável por inserir linha em branco na interface da tabela Equipe.
public void inserirLinhaEquipe (JTable jTable)
{
    ime.criaLinhaTabelaEquipe(jtable);
}

//Método responsável por inserir linha em branco na interface da tabela Fase.
public void inserirLinhaFase (JTable jTable)
{
    imf.criaLinhaTabelaFase(jtable);
}

//Método responsável por abrir a caixa de diálogo informando que o usuário
//não pode incluir linhas na tabela Atividade.
public void inserirLinhaAtividade (JTable jTable)
{
    JOptionPane.showMessageDialog(null, "A tabela Atividade não permite
    inclusões!",
        "Inserir", JOptionPane.INFORMATION_MESSAGE);
}

//Método responsável por inserir linha em branco na interface da tabela Host.
public void inserirLinhaHost (JTable jTable)
{
    imh.criaLinhaTabelaHost(jtable);
}

//Método responsável por inserir linha em branco na interface da tabela Serviço.
public void inserirLinhaServico (JTable jTable)
{
    imserv.criaLinhaTabelaServico(jtable);
}

```

```

}

//Método responsável por abrir a caixa de diálogo informando que o usuário
//não pode incluir linhas na tabela Monitoramento.
public void inserirLinhaMonitoramento (JTable jtable)
{
    JOptionPane.showMessageDialog(null, "A tabela Monitoramento não permite
    inclusões!",
        "Inserir", JOptionPane.INFORMATION_MESSAGE);
}
//Método responsável por atualizar a tabela Sistema no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDSistema(JTable jtable)
{
    Sistema[] s;

    try {
        SistemaDAO sistemaDAO = new SistemaDAO();

        if (ims.quantRegistrosNovos(jtable) > 0)
        {
            s = new Sistema[ims.quantRegistrosNovos(jtable)];
            for (int i = 0; i < s.length; i++) s[i] = new Sistema();
            ims.preencheRegistrosNovos (s, jtable);
            for (int i = 0; i < s.length; i++) sistemaDAO.save(s[i]);
        }
        s = new Sistema[ims.quantRegistrosAntigos(jtable)];
        for (int i = 0; i < s.length; i++) s[i] = new Sistema();
        ims.preencheRegistrosAntigos (s, jtable);
        for (int i = 0; i < s.length; i++) sistemaDAO.update(s[i]);
        JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
            "Atualizar", JOptionPane.INFORMATION_MESSAGE);
        consultarSistema (jtable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por atualizar a tabela Equipe no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDEquipe(JTable jtable)
{
    Equipe[] e;

    try {
        EquipeDAO equipeDAO = new EquipeDAO();

        if (ime.quantRegistrosNovos(jtable) > 0)
        {
            e = new Equipe[ime.quantRegistrosNovos(jtable)];
            for (int i = 0; i < e.length; i++) e[i] = new Equipe();
            ime.preencheRegistrosNovos (e, jtable);
            for (int i = 0; i < e.length; i++) equipeDAO.save(e[i]);
            //falta implementar checagem do código da fase
        }
        e = new Equipe[ime.quantRegistrosAntigos(jtable)];
        for (int i = 0; i < e.length; i++) e[i] = new Equipe();
        ime.preencheRegistrosAntigos (e, jtable);
        for (int i = 0; i < e.length; i++) equipeDAO.update(e[i]);
        //falta implementar checagem do código da fase
        JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
            "Atualizar", JOptionPane.INFORMATION_MESSAGE);
        consultarEquipe (jtable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

```

```

}

//Método responsável por atualizar a tabela Fase no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDFase(JTable jtable)
{
    Fase[] f;

    try {
        FaseDAO faseDAO = new FaseDAO();

        if (imf.quantRegistrosNovos(jtable) > 0)
        {
            f = new Fase[imf.quantRegistrosNovos(jtable)];
            for (int i = 0; i < f.length; i++) f[i] = new Fase();
            imf.preencheRegistrosNovos (f, jtable);
            for (int i = 0; i < f.length; i++) faseDAO.save(f[i]);
        }
        f = new Fase[imf.quantRegistrosAntigos(jtable)];
        for (int i = 0; i < f.length; i++) f[i] = new Fase();
        imf.preencheRegistrosAntigos (f, jtable);
        for (int i = 0; i < f.length; i++) faseDAO.update(f[i]);
        JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
            "Atualizar", JOptionPane.INFORMATION_MESSAGE);
        consultarFase (jtable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por atualizar a tabela Atividade no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDAtividade(JTable jtable)
{
    Atividade[] a;

    try {
        AtividadeDAO atividadeDAO = new AtividadeDAO();

        a = new Atividade[ima.quantRegistrosAntigos(jtable)];
        for (int i = 0; i < a.length; i++) a[i] = new Atividade();
        ima.preencheRegistrosAntigos (a, jtable);
        for (int i = 0; i < a.length; i++) atividadeDAO.update(a[i]);
        JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
            "Atualizar", JOptionPane.INFORMATION_MESSAGE);
        consultarAtividade (jtable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por atualizar a tabela Host no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDHost(JTable jtable)
{
    Host[] h;

    try {
        HostDAO hostDAO = new HostDAO();

        if (imh.quantRegistrosNovos(jtable) > 0)
        {
            h = new Host[imh.quantRegistrosNovos(jtable)];
            for (int i = 0; i < h.length; i++) h[i] = new Host();
            imh.preencheRegistrosNovos (h, jtable);
            for (int i = 0; i < h.length; i++) hostDAO.save(h[i]);
        }
    }
}

```

```

    }
    h = new Host[imh.quantRegistrosAntigos(jtable)];
    for (int i = 0; i < h.length; i++) h[i] = new Host();
    imh.preencheRegistrosAntigos (h, jtable);
    for (int i = 0; i < h.length; i++) hostDAO.update(h[i]);
    JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
        "Atualizar", JOptionPane.INFORMATION_MESSAGE);
    consultarHost (jtable);
}
catch (SQLException e1) {
    e1.printStackTrace();
}
}

//Método responsável por atualizar a tabela Serviço no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void atualizarBDServico(JTable jtable)
{
    Servico[] serv;

    try {
        ServicoDAO servicoDAO = new ServicoDAO();

        if (imserv.quantRegistrosNovos(jtable) > 0)
        {
            serv = new Servico[imserv.quantRegistrosNovos(jtable)];
            for (int i = 0; i < serv.length; i++) serv[i] = new Servico();
            imserv.preencheRegistrosNovos (serv, jtable);
            for (int i = 0; i < serv.length; i++) servicoDAO.save(serv[i]);
        }
        serv = new Servico[imserv.quantRegistrosAntigos(jtable)];
        for (int i = 0; i < serv.length; i++) serv[i] = new Servico();
        imserv.preencheRegistrosAntigos (serv, jtable);
        for (int i = 0; i < serv.length; i++) servicoDAO.update(serv[i]);
        JOptionPane.showMessageDialog(null, "Dados atualizados com sucesso!",
            "Atualizar", JOptionPane.INFORMATION_MESSAGE);
        consultarServico (jtable);
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por abrir a caixa de diálogo informando que o usuário
//não pode incluir linhas na tabela Monitoramento.
public void atualizarBDMonitoramento (JTable jtable)
{
    JOptionPane.showMessageDialog(null, "A tabela Monitoramento não permite
    alterações!",
        "Atualizar", JOptionPane.INFORMATION_MESSAGE);
}

//Método responsável por excluir linha na tabela Sistema no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void excluirSistema(JTable jtable)
{
    Sistema s;

    try {
        SistemaDAO sistemaDAO = new SistemaDAO();

        if (ims.existeLinhaSelecionada(jtable))
        {
            s = new Sistema();
            ims.getRegistroSelecionado (s, jtable);
            sistemaDAO.remove(s);
            consultarSistema (jtable);
        }
    }
}

```

```

        catch (SQLException e1) {
            e1.printStackTrace();
        }
    }

    //Método responsável por excluir linha na tabela Equipe no banco de dados
    //e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
    public void excluirEquipe(JTable jtable)
    {
        Equipe e;

        try {
            EquipeDAO equipeDAO = new EquipeDAO();

            if (ime.existeLinhaSelecioneada(jtable))
            {
                e = new Equipe();
                ime.getRegistroSelecioneado (e, jtable);
                equipeDAO.remove(e);
                consultarEquipe (jtable);
            }
        }
        catch (SQLException e1) {
            e1.printStackTrace();
        }
    }

    //Método responsável por excluir linha na tabela Fase no banco de dados
    //e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
    public void excluirFase(JTable jtable)
    {
        Fase f;

        try {
            FaseDAO faseDAO = new FaseDAO();

            if (imf.existeLinhaSelecioneada(jtable))
            {
                f = new Fase();
                imf.getRegistroSelecioneado (f, jtable);
                faseDAO.remove(f);
                consultarFase (jtable);
            }
        }
        catch (SQLException e1) {
            e1.printStackTrace();
        }
    }

    //Método responsável por abrir a caixa de diálogo informando que o usuário
    //não pode excluir linhas na tabela Atividade.
    public void excluirAtividade(JTable jtable)
    {
        JOptionPane.showMessageDialog(null, "A tabela Atividade não permite
        exclusões!",
        "Excluir", JOptionPane.INFORMATION_MESSAGE);
    }

    //Método responsável por excluir linha na tabela Host no banco de dados
    //e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
    public void excluirHost(JTable jtable)
    {
        Host h;

        try {
            HostDAO hostDAO = new HostDAO();

```

```

        if (imh.existeLinhaSelecioneada(jtable))
        {
            h = new Host();
            imh.getRegistroSelecioneado (h, jtable);
            hostDAO.remove(h);
            consultarHost (jtable);
        }
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por excluir linha na tabela Serviço no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void excluirServico(JTable jtable)
{
    Servico serv;

    try {
        ServicoDAO servicoDAO = new ServicoDAO();

        if (imserv.existeLinhaSelecioneada(jtable))
        {
            serv = new Servico();
            imserv.getRegistroSelecioneado (serv, jtable);
            servicoDAO.remove(serv);
            consultarServico (jtable);
        }
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}

//Método responsável por excluir linha na tabela Monitoramento no banco de dados
//e em seguida, exibir todo o conteúdo dessa tabela na janela da interface.
public void excluirMonitoramento(JTable jtable)
{
    Monitoramento m;

    try {
        MonitoramentoDAO monitoramentoDAO = new MonitoramentoDAO();

        if (imm.existeLinhaSelecioneada(jtable))
        {
            m = new Monitoramento();
            imm.getRegistroSelecioneado (m, jtable);
            monitoramentoDAO.remove(m);
            consultarMonitoramento (jtable);
        }
    }
    catch (SQLException e1) {
        e1.printStackTrace();
    }
}
}
}

Classe MonitorNodos.java

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */

```

```

/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável pelo controle da monitoração dos nodos. */
/*****

```

```
package controle;
```

```
public class MonitorNodos {
```

```
    boolean pararMonitoramento = true;
```

```
    boolean AcheiProbSeg = false;
```

```
    public boolean isPararMonitoramento() {
```

```
        return pararMonitoramento;
```

```
    }
```

```
    public void setPararMonitoramento(boolean pararMonitoramento) {
```

```
        this.pararMonitoramento = pararMonitoramento;
```

```
    }
```

```
    public boolean isAcheiProbSeg() {
```

```
        return AcheiProbSeg;
```

```
    }
```

```
    public void setAcheiProbSeg(boolean acheiProbSeg) {
```

```
        AcheiProbSeg = acheiProbSeg;
```

```
    }
```

```
    }
```

Classe *MonitorNodosThread.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável pelo processo de monitoramento dos nodos que */
/* executa de forma concorrente com o resto do gestor. */
/*****

```

```
package controle;
```

```
import javax.swing.JLabel;
```

```
import javax.swing.JTextPane;
```

```
import javax.swing.JOptionPane;
```

```
import modelo.*;
```

```
import DAO.*;
```

```
import interface_rede.*;
```

```
public class MonitorNodosThread extends Thread{
```

```
    JLabel jlabelmon;
```

```
    JTextPane jtextpane;
```

```
    MonitorNodos mn;
```

```
    public MonitorNodosThread (JLabel jlabelmon, JTextPane jtextpane, MonitorNodos mn)
```



```

{
    this.jlabelmon = jlabelmon;
    this.jtextpane = jtextpane;
    this.mn = mn;
}

//Este método retorna o número da porta de um dado host que esteja funcionando como
Fase,
//ou seja, um Nodo.
public int getPortaHostFase(Servico[] servicos, String IPhost) {

    for (int i = 0; i < servicos.length; i++)
    {
        if (servicos[i].getStatus().equals("Fase")
            && servicos[i].getHost().equals(IPhost))
            return servicos[i].getPorta();
    }
    return 0;
}

//Este método retorna o atraso relativo, ou seja, o atraso medido do nodo comparado
com o anterior
//que foi obtido pela calibração o qual se encontra cadastrado no banco.
public double ajustaAtraso (double AtrasoAbsoluto, String IPhost)
{
    double AtrasoAjustado = 10;
    try
    {
        HostDAO hdao = new HostDAO();
        Host h = hdao.retrieveByIP(IPhost);
        AtrasoAjustado = AtrasoAbsoluto/h.getAtraso();
        if (AtrasoAjustado > 10) AtrasoAjustado = 10;
        if (AtrasoAjustado < 0) AtrasoAjustado = 10;
    } catch (Exception e) {
        System.err.println(e.toString());
    }
    return AtrasoAjustado;
}

//Este método calcula a quantidade de serviços sem acesso.
public int ajustaServicos (int ServicosAcessados, String IPhost)
{
    int ServicosSemAcesso = 4;
    try
    {
        HostDAO hdao = new HostDAO();
        Host h = hdao.retrieveByIP(IPhost);
        ServicosSemAcesso = h.getServicos() - ServicosAcessados;
        if (ServicosSemAcesso < 0) ServicosSemAcesso = 0;
        if (ServicosSemAcesso > 4) ServicosSemAcesso = 4;
    } catch (Exception e) {
        System.err.println(e.toString());
    }
    return ServicosSemAcesso;
}

//Este método calcula a quantidade relativa de artefatos atrasados, ou seja, o
percentual
//de artefatos atrasados em relação ao total de artefatos sob análise.
public int artefatosAtrasadosRelativo (String IPhost)
{
    int artatrasados = 0;
    int arterrados = 0;
    int total = 0;
    int artrel = 0;
    try
    {
        AtividadeDAO adao = new AtividadeDAO();
        Atividade[] atividades = adao.retrieveByHost(IPhost);

```

```

        for (int i = 0; i < atividades.length; i++)
        {
            if (atividades[i].getStatus().equals("Atrasado") ||
                atividades[i].getStatus().equals("Atrasado e Errado"))
artatrasados++;
            if (atividades[i].getStatus().equals("Errado") ||
                atividades[i].getStatus().equals("Atrasado e Errado"))
arterrados++;
            if (!atividades[i].getStatus().equals("Validando") &&
                !atividades[i].getStatus().equals("Analisado")) total++;
        }
        if (total > 0)
            artrel = (int) (100*(double)artatrasados/(double)total);
    } catch (Exception e) {
        System.err.println(e.toString());
    }
    return artrel;
}

//Este método calcular a quantidade relativa de artefatos errados.
public int artefatosErradosRelativo (String IPhost)
{
    int artatrasados = 0;
    int arterrados = 0;
    int total = 0;
    int artrel = 0;
    try
    {
        AtividadeDAO adao = new AtividadeDAO();
        Atividade[] atividades = adao.retrieveByHost(IPhost);
        for (int i = 0; i < atividades.length; i++)
        {
            if (atividades[i].getStatus().equals("Atrasado") ||
                atividades[i].getStatus().equals("Atrasado e Errado"))
artatrasados++;
            if (atividades[i].getStatus().equals("Errado") ||
                atividades[i].getStatus().equals("Atrasado e Errado"))
arterrados++;
            if (!atividades[i].getStatus().equals("Validando") &&
                !atividades[i].getStatus().equals("Analisado")) total++;
        }
        if (total > 0)
            artrel = (int) (100*(double)arterrados/(double)total);
    } catch (Exception e) {
        System.err.println(e.toString());
    }
    return artrel;
}

//Método responsável por iniciar o monitoramento dos serviços.
public void iniciarMonitoramento ()
{
    int contagem = 0;
    Cliente cliente = new Cliente();
    DeFuzzyBean dfbean = new DeFuzzyBean();
    String msg;

do {
    contagem = 0;
    while ((!this.mn.isPararMonitoramento()) && (!this.mn.isAcheiProbSeg())) {
        contagem++;
        try {
            Servico[] servicos;
            ServicoDAO servicoDAO = new ServicoDAO();
            servicos = servicoDAO.retrieveAll();
            MonitoramentoDAO mondao = new MonitoramentoDAO();

            FaseDAO fdao = new FaseDAO();

```

```

Fase[] fases = fdao.retrieveAll();
Monitoramento mon = new Monitoramento();
for (int i = 0; i < fases.length; i++)
{
    jlabelmon.setText
    ("MONITORANDO HOST " + fases[i].getHost() + " - Iteração
" + contagem);
    mon.setHost(fases[i].getHost());

    mon.setServicos(ajustaServicos(cliente.getServicos(fases[i].getHost(),
servicos),fases[i].getHost()));
    msg = "Serviços sem acesso = " + mon.getServicos();
    jtextpane.setText(msg);

    mon.setAtraso(ajustaAtraso(cliente.getAtraso(fases[i].getHost(),
getPortaHostFase(servicos,fases[i].getHost()),
fases[i].getHost()));
    msg = msg + "\n" + "Atraso = " + mon.getAtraso();
    jtextpane.setText(msg);

    mon.setTentativas(cliente.getTentativas(fases[i].getHost()));
    msg = msg + "\n" + "Tentativas = " + mon.getTentativas();
    jtextpane.setText(msg);
    dfbean.setCod_sistema(1); // SIR
    dfbean.setCod_variavel1(1); // código do Atraso
    dfbean.setValor_var1((int)mon.getAtraso());
    dfbean.setCod_variavel2(2); // código do
Numero_Tentativas
    dfbean.setValor_var2(mon.getTentativas());
    dfbean.setCod_variavel3(3); // código do
Servicos_sem_Acesso
    dfbean.setValor_var3(mon.getServicos());
    dfbean.setResultadoFinal();
    mon.setIndiceProbRede(dfbean.getResultadoFinal());
    msg = msg + "\n" + "Índice de Problemas na Rede = " +
mon.getIndiceProbRede();
    jtextpane.setText(msg);

    mon.setArtefatosAtrasados(artefatosAtrasadosRelativo(fases[i].getHost()));
    msg = msg + "\n" + "Artefatos Atrasados = " +
mon.getArtefatosAtrasados();
    jtextpane.setText(msg);

    mon.setArtefatosErrados(artefatosErradosRelativo(fases[i].getHost()));
    msg = msg + "\n" + "Artefatos com Erro = " +
mon.getArtefatosErrados();
    jtextpane.setText(msg);
    dfbean.setCod_sistema(2); // SIS
    dfbean.setCod_variavel1(5); // código do
Indice_Problemas_Rede
    dfbean.setValor_var1((int)mon.getIndiceProbRede());
    dfbean.setCod_variavel2(6); // código do
Artefatos_Fora_do_Prazo
    dfbean.setValor_var2(mon.getArtefatosAtrasados());
    dfbean.setCod_variavel3(7); // código do
Artefatos_Com_Erro
    dfbean.setValor_var3(mon.getArtefatosErrados());
    dfbean.setResultadoFinal();
    mon.setIndiceProbSeg(dfbean.getResultadoFinal());
    msg = msg + "\n" + "Índice de Problemas de Segurança = "
+ mon.getIndiceProbSeg();
    jtextpane.setText(msg);
    mondao.save(mon);
    if (mon.getIndiceProbSeg() > 60)
    {
        this.mn.setAcheiProbSeg(true);
    }
}

```

```

                                JOptionPane.showMessageDialog
                                (null, "Problema de segurança no host " +
fases[i].getHost() + "!",
                                "Monitoramento",
JOptionPane.INFORMATION_MESSAGE);
                                }
                                Thread.sleep(3000);
                                } catch (InterruptedException e) {}
                                catch (Exception e) {
                                System.err.println(e.toString());
                                }
                                }
                                jlabelmon.setVisible(false);
                                jtextpane.setVisible(false);
                                jlabelmon.setText(" ");
                                jtextpane.setText(" ");
                                }
                                while (true);
                                }

public void run ()
{
    iniciarMonitoramento ();
}
}

```

A.2 PACOTE DAO

Classe *AtividadeDAO.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe busca os dados no banco e atualiza a interface da Tabela */
/* Atividade. */
*****/
package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Atividade;
import controle.Conexao;

public class AtividadeDAO {

    private Connection con;
    private Statement st = null;
    private ResultSet rs = null;

    // Inicializa a conexão no construtor
    public AtividadeDAO() throws SQLException {
        con = Conexao.getConnection();
        st = con.createStatement();
    }
}

```

```

//Método que exclui um registro da tabela Atividade do banco.
public void remove(Atividade atividade) {

try {
    // Monta a string sql
    String sql = "delete from atividade where codigo = ";
    sql = sql + atividade.getCodigo();
    int res = st.executeUpdate (sql);
    if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                "Excluir",
JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                + e, "Excluir",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que consulta todos os registros da tabela Atividade do banco.
public Atividade[] retrieveAll() {

    Atividade[] aux;
    Atividade[] aux1 = new Atividade[1];

try {
    rs = st.executeQuery("select count(codigo) from atividade");
    rs.next();
    int i = rs.getInt(1);

    rs = st.executeQuery("select * from atividade order by codigo");
    aux = new Atividade[i];

    while(rs.next())
    {
        int j= rs.getRow()-1;
        aux[j] = new Atividade();
        aux[j].setCodigo(rs.getInt("codigo"));
        aux[j].setHost(rs.getString("host"));
        aux[j].setStatus(rs.getString("status"));
        aux[j].setNomeArqArt(rs.getString("nome_arquivo_artefato"));
    }
    return aux;

} catch (SQLException e) {
    // Retorna uma mensagem informando o erro

        JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
dados!\nInformações sobre o erro:"
                                                + e, "Consultar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
    return aux1;
}

//Método que consulta todos os registros da tabela Atividade do banco
//referentes a um host específico.
public Atividade[] retrieveByHost(String host) {

    Atividade[] aux;

try {
    rs = st.executeQuery

```

```

        ("select count(codigo) from atividade where host = '" +
host + "'");
        rs.next();
        int i = rs.getInt(1);
        rs = st.executeQuery("select * from atividade where host = '" +
host + "'");
        aux = new Atividade[i];
        while (rs.next())
        {
            int j= rs.getRow()-1;
            aux[j] = new Atividade();
            aux[j].setCodigo(rs.getInt("codigo"));
            aux[j].setHost(rs.getString("host"));
            aux[j].setStatus(rs.getString("status"));

            aux[j].setNomeArqArt(rs.getString("nome_arquivo_artefato"));
        }
        return aux;

    } catch (SQLException e) {
        return null;
    }
}

//Método que atualiza um registro da tabela Atividade do banco.
public void update(Atividade atividade) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "update atividade set host = ?, status = ?,
nome_arquivo_artefato = ?";
        sql = sql + " where codigo = ?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, atividade.getHost());
        pstmt.setString(2, atividade.getStatus());
        pstmt.setString(3, atividade.getNomeArqArt());
        pstmt.setInt(4, atividade.getCodigo());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
dados!\nInformações sobre o erro:"
+ e, "Atualizar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que insere um registro na tabela Atividade do banco.
public void save(Atividade atividade) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql

```

```

        String sql = "insert into atividade (host, status,
nome_arquivo_artefato) values(?,?,?)";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, atividade.getHost());
        pstmt.setString(2, atividade.getStatus());
        pstmt.setString(3, atividade.getNomeArqArt());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
                                     + e, "Salvar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
}
}

```

Classe *EquipeDAO.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe busca os dados no banco e atualiza a interface da Tabela Equipe */
*****/

package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Equipe;
import controle.Conexao;

public class EquipeDAO {

    private Connection con;
    private Statement st = null;
    private ResultSet rs = null;

    // Inicializa a conexão no construtor
    public EquipeDAO() throws SQLException {
        con = Conexao.getConnection();
        st = con.createStatement();
    }

    //Método que exclui um registro da tabela Equipe do banco.
    public void remove(Equipe equipe) {

        try {

            // Monta a string sql
            String sql = "delete from equipe where codigo_funcionario = ";

```

```

        sql = sql + equipe.getCodigo_funcionario();
        int res = st.executeUpdate (sql);
        if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                "Excluir",
JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                + e, "Excluir",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que consulta todos os registros da tabela Equipe do banco.
public Equipe[] retrieveAll() {

    Equipe[] aux;
    Equipe[] aux1 = new Equipe[1];

    try {
        rs = st.executeQuery("select count(codigo_funcionario) from equipe");
        rs.next();
        int i = rs.getInt(1);

        rs = st.executeQuery("select * from equipe order by
codigo_funcionario");
        aux = new Equipe[i];

        while(rs.next())
        {
            int j= rs.getRow()-1;
            aux[j] = new Equipe();
            aux[j].setCodigo_funcionario(rs.getInt("codigo_funcionario"));
            aux[j].setNome_funcionario(rs.getString("nome_funcionario"));
            aux[j].setCargo_funcionario(rs.getString("cargo_funcionario"));
            aux[j].setCodigo_fase(rs.getInt("codigo_fase"));

        }
        return aux;

    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro

        JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
dados!\nInformações sobre o erro:"
                                                + e, "Consultar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
    return aux1;
}
/*
public Equipe retrieveByID(String nome) {
    return null;
}
*/
//Método que atualiza um registro da tabela Equipe do banco.
public void update(Equipe equipe) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {

```



```

        // Monta a string sql
        String sql = "update equipe set nome_funcionario = ?,
cargo_funcionario = ?, codigo_fase = ?";
        sql = sql + " where codigo_funcionario = ?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, equipe.getNome_funcionario());
        pstmt.setString(2, equipe.getCargo_funcionario());
        pstmt.setInt(3, equipe.getCodigo_fase());
        pstmt.setInt(4, equipe.getCodigo_funcionario());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
dados!\nInformações sobre o erro:"
                                + e, "Atualizar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que insere um registro na tabela Equipe do banco.
public void save(Equipe equipe) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "insert into equipe (nome_funcionario, cargo_funcionario,
codigo_fase) values(?,?,?)";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, equipe.getNome_funcionario());
        pstmt.setString(2, equipe.getCargo_funcionario());
        pstmt.setInt(3, equipe.getCodigo_fase());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
                                + e, "Salvar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
}
}

```

Classe *FaseDAO.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */

```

```

/* Dissertação de Mestrado
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação
/* em nuvem
/* Orientador: Nival Nunes de Almeida
/* Coorientador: Orlando Bernardo Filho
/*
/* Esta classe busca os dados no banco e atualiza a interface da Tabela Fase
/*****
package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Fase;
import controle.Conexao;

public class FaseDAO {

    private Connection con;
    private Statement st = null;
    private ResultSet rs = null;

    // Inicializa a conexão no construtor
    public FaseDAO() throws SQLException {
        con = Conexao.getConnection();
        st = con.createStatement();
    }

    //Método que exclui um registro da tabela Fase do banco.
    public void remove(Fase fase) {

        try {
            // Monta a string sql
            String sql = "delete from fase where codigo_fase = ";
            sql = sql + fase.getCodigo();
            int res = st.executeUpdate (sql);
            if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                    "Excluir",
JOptionPane.INFORMATION_MESSAGE);
        } catch (SQLException e) {
            // Retorna uma mensagem informando o erro
            JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                    + e, "Excluir",
JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }

    //Método que consulta todos os registros da tabela Fase do banco.
    public Fase[] retrieveAll() {

        Fase[] aux;
        Fase[] aux1 = new Fase[1];

        try {
            rs = st.executeQuery("select count(codigo_fase) from fase");
            rs.next();
            int i = rs.getInt(1);

            rs = st.executeQuery("select * from fase order by codigo_fase");
            aux = new Fase[i];

            while(rs.next())
            {
                int j= rs.getRow()-1;

```

```

        aux[j] = new Fase();
        aux[j].setCodigo(rs.getInt("codigo_fase"));
        aux[j].setNome(rs.getString("nome_fase"));
        aux[j].setHost(rs.getString("host"));
        aux[j].setStatus(rs.getString("status"));
        aux[j].setDataInicio(rs.getDate("data_inicio"));
        aux[j].setDataFim(rs.getDate("data_final"));
        aux[j].setCodigoSistema(rs.getInt("codigo_sistema"));

    }
    return aux;

} catch (SQLException e) {
    // Retorna uma mensagem informando o erro

    JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
    dados!\nInformações sobre o erro:"
                                + e, "Consultar",
    JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
}
return aux1;
}

//Método que consulta todos os registros da tabela Fase do banco
//referentes a um host específico.
public Fase retrieveByHost(String host) {

    Fase aux = new Fase();

    try {
        rs = st.executeQuery("select * from fase where host = '" + host
+ "'");
        if (rs.next())
        {
            aux.setCodigo(rs.getInt("codigo_fase"));
            aux.setNome(rs.getString("nome_fase"));
            aux.setHost(rs.getString("host"));
            aux.setStatus(rs.getString("status"));
            aux.setDataInicio(rs.getDate("data_inicio"));
            aux.setDataFim(rs.getDate("data_final"));
            aux.setCodigoSistema(rs.getInt("codigo_sistema"));
        }
        return aux;

    } catch (SQLException e) {
        return null;
    }
}

//Método que atualiza um registro da tabela Fase do banco.
public void update(Fase fase) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "update fase set nome_fase = ?, host = ?, status = ?,
data_inicio = ?, ";
        sql = sql + "data_final = ?, codigo_sistema = ? where codigo_fase =
?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

```

```

// Coloca os verdadeiros valores no lugar dos ?
pstm.setString(1, fase.getNome());
pstm.setString(2, fase.getHost());
pstm.setString(3, fase.getStatus());
pstm.setDate(4, fase.getDataInicio());
pstm.setDate(5, fase.getDataFim());
pstm.setInt(6, fase.getCodigoSistema());
pstm.setInt(7, fase.getCodigo());
// Executa
pstm.execute();
} catch (SQLException e) {
// Retorna uma mensagem informando o erro
JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
dados!\nInformações sobre o erro:"
+ e, "Atualizar",
JOptionPane.ERROR_MESSAGE);
e.printStackTrace();
}
}

//Método que insere um registro na tabela Fase do banco.
public void save(Fase fase) {

// Pega a conexão estabelecida
Connection conn = this.con;
// Cria um PreparedStatement
PreparedStatement pstm = null;

try {
// Monta a string sql
String sql = "insert into fase (nome_fase, host, status, data_inicio,
data_final, " +
"codigo_sistema) values(?,?,?,?,?,?)";

// Passa a string para o PreparedStatement
pstm = conn.prepareStatement(sql);

// Coloca os verdadeiros valores no lugar dos ?
pstm.setString(1, fase.getNome());
pstm.setString(2, fase.getHost());
pstm.setString(3, fase.getStatus());
pstm.setDate(4, fase.getDataInicio());
pstm.setDate(5, fase.getDataFim());
pstm.setInt(6, fase.getCodigoSistema());
// Executa
pstm.execute();
} catch (SQLException e) {
// Retorna uma mensagem informando o erro
JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
+ e, "Salvar",
JOptionPane.ERROR_MESSAGE);
e.printStackTrace();
}
}
}

```

Classe *HostDAO.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */

```

```

/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe busca os dados no banco e atualiza a interface da Tabela Host */
/*****/
package DAO;

import java.sql.*;

import javax.swing.JOptionPane;
import modelo.Host;
import controle.Conexao;

public class HostDAO {

    private Connection con;
    private Statement st = null;
    private ResultSet rs = null;

    // Inicializa a conexão no construtor
    public HostDAO() throws SQLException {
        con = Conexao.getConnection();
        st = con.createStatement();
    }

    //Método que exclui um registro da tabela Host do banco.
    public void remove(Host host) {

        try {
            // Monta a string sql
            String sql = "delete from host where endereco_ip = ";
            sql = sql + host.getIP() + " ";
            int res = st.executeUpdate (sql);
            if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                    "Excluir",
JOptionPane.INFORMATION_MESSAGE);
        } catch (SQLException err) {
            // Retorna uma mensagem informando o erro
            JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                    + err, "Excluir",
JOptionPane.ERROR_MESSAGE);
            err.printStackTrace();
        }
    }

    //Método que consulta todos os registros da tabela Host do banco.
    public Host[] retrieveAll() {

        Host[] aux;
        Host[] aux1 = new Host[1];

        try {
            rs = st.executeQuery("select count(endereco_ip) from host");
            rs.next();
            int i = rs.getInt(1);

            rs = st.executeQuery("select * from host");
            aux = new Host[i];

            while(rs.next())
            {
                int j= rs.getRow()-1;
                aux[j] = new Host();
                aux[j].setIP(rs.getString("endereco_ip"));
            }
        }
    }
}

```

```

        aux[j].setServicos(rs.getInt("servicos"));
        aux[j].setAtraso(rs.getDouble("atraso"));

    }
    return aux;

} catch (SQLException err) {
    // Retorna uma mensagem informando o erro

    JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
    dados!\nInformações sobre o erro:"
                                + err, "Consultar",
    JOptionPane.ERROR_MESSAGE);
    err.printStackTrace();
}
return aux1;
}

//Método que consulta todos os registros da tabela Host do banco
//referentes a um host específico.
public Host retrieveByIP(String IP) {

    Host aux = new Host();

    try {
        rs = st.executeQuery("select * from host where endereco_ip = '"
+ IP + "'");
        if (rs.next())
        {
            aux.setIP(rs.getString("endereco_ip"));
            aux.setServicos(rs.getInt("servicos"));
            aux.setAtraso(rs.getDouble("atraso"));
        }
        return aux;

    } catch (SQLException e) {
        return null;
    }
}

//Método que atualiza um registro da tabela Host do banco.
public void update(Host host) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "update host set servicos = ?, atraso = ?";
        sql = sql + " where endereco_ip = ?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setInt(1, host.getServicos());
        pstmt.setDouble(2, host.getAtraso());
        pstmt.setString(3, host.getIP());
        // Executa
        pstmt.execute();
    } catch (SQLException err) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
        dados!\nInformações sobre o erro:"
                                    + err, "Atualizar",
        JOptionPane.ERROR_MESSAGE);
    }
}

```

```

        err.printStackTrace();
    }
}

//Método que insere um registro na tabela Host do banco.
public void save(Host host) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "insert into host (endereco_ip, servicos, atraso)
values(?,?,?)";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, host.getIP());
        pstmt.setInt(2, host.getServicos());
        pstmt.setDouble(3, host.getAtraso());
        // Executa
        pstmt.execute();
    } catch (SQLException err) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
                                     + err, "Salvar",
JOptionPane.ERROR_MESSAGE);
        err.printStackTrace();
    }
}
}

```

Classe *MonitoramentoDAO.java*

```

/*****
/* Gestor de Software na Nuvem
/* Autor: Carolina Y. Ji
/* Universidade do Estado do Rio de Janeiro
/* Programa de Pós Graduação em Engenharia Eletrônica
/* Dissertação de Mestrado
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação
/* em nuvem
/* Orientador: Nival Nunes de Almeida
/* Coorientador: Orlando Bernardo Filho
/*
/* Esta classe busca os dados no banco e atualiza a interface da Tabela
/* Monitoramento
*****/

package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Monitoramento;
import controle.Conexao;

public class MonitoramentoDAO {

    private Connection con;
    private Statement st = null;

```

```

private ResultSet rs = null;

// Inicializa a conexão no construtor
public MonitoramentoDAO() throws SQLException {
    con = Conexao.getConnection();
    st = con.createStatement();
}

//Método que exclui um registro da tabela Monitoramento do banco.
public void remove(Monitoramento monitoramento) {

    try {
        // Monta a string sql
        String sql = "delete from monitoramento where codigo = ";
        sql = sql + monitoramento.getCodigo();
        int res = st.executeUpdate (sql);
        if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                "Excluir",
JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                + e, "Excluir",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que consulta todos os registros da tabela Monitoramento do banco.
public Monitoramento[] retrieveAll() {

    Monitoramento[] aux;
    Monitoramento[] aux1 = new Monitoramento[1];

    try {
        rs = st.executeQuery("select count(codigo) from monitoramento");
        rs.next();
        int i = rs.getInt(1);

        rs = st.executeQuery("select * from monitoramento order by codigo");
        aux = new Monitoramento[i];

        while(rs.next())
        {
            int j= rs.getRow()-1;
            aux[j] = new Monitoramento();
            aux[j].setCodigo(rs.getInt("codigo"));
            aux[j].setHost(rs.getString("host"));
            aux[j].setAtraso(rs.getDouble("atraso"));
            aux[j].setServicos(rs.getInt("servicos"));
            aux[j].setTentativas(rs.getInt("tentativas"));
            aux[j].setIndiceProbRede(rs.getDouble("indice_prob_rede"));
            aux[j].setArtefatosErrados(rs.getInt("artefatos_errados"));
            aux[j].setArtefatosAtrasados(rs.getInt("artefatos_atrasados"));

            aux[j].setIndiceProbSeg(rs.getDouble("indice_prob_seg"));
        }
        return aux;
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro

        JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
dados!\nInformações sobre o erro:"
                                                + e, "Consultar",
JOptionPane.ERROR_MESSAGE);
    }
}

```



```

        e.printStackTrace();
    }
    return aux1;
}
/*
public Monitoramento retrieveByID(String nome) {
    return null;
}
*/

//Método que atualiza um registro da tabela Monitoramento do banco.
public void update(Monitoramento monitoramento) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "update monitoramento set host = ?, atraso = ?, servicos
= ?, tentativas = ?, ";
        sql = sql + "indice_prob_rede = ?, artefatos_errados = ?,
artefatos_atrasados = ?, ";
        sql = sql + "indice_prob_seg = ? where codigo = ?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, monitoramento.getHost());
        pstmt.setDouble(2, monitoramento.getAtraso());
        pstmt.setInt(3, monitoramento.getServicos());
        pstmt.setInt(4, monitoramento.getTentativas());
        pstmt.setDouble(5, monitoramento.getIndiceProbRede());
        pstmt.setInt(6, monitoramento.getArtefatosErrados());
        pstmt.setInt(7, monitoramento.getArtefatosAtrasados());
        pstmt.setDouble(8, monitoramento.getIndiceProbSeg());
        pstmt.setInt(9, monitoramento.getCodigo());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
dados!\nInformações sobre o erro:"
+ e, "Atualizar",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

//Método que insere um registro na tabela Monitoramento do banco.
public void save(Monitoramento monitoramento) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "insert into monitoramento (host, atraso, servicos, tentativas,
indice_prob_rede, " +
        "artefatos_errados, artefatos_atrasados, indice_prob_seg)
values(?,?,?,?,?,?,?,?)";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

```

```

// Coloca os verdadeiros valores no lugar dos ?
    pstmt.setString(1, monitoramento.getHost());
    pstmt.setDouble(2, monitoramento.getAtraso());
    pstmt.setInt(3, monitoramento.getServicos());
    pstmt.setInt(4, monitoramento.getTentativas());
    pstmt.setDouble(5, monitoramento.getIndiceProbRede());
    pstmt.setInt(6, monitoramento.getArtefatosErrados());
    pstmt.setInt(7, monitoramento.getArtefatosAtrasados());
    pstmt.setDouble(8, monitoramento.getIndiceProbSeg());

// Executa
    pstmt.execute();
} catch (SQLException e) {
// Retorna uma mensagem informando o erro
    JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
                                + e, "Salvar",
JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
}
}
}

```

Classe ServicoDAO.java

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe busca os dados no banco e atualiza a interface da Tabela */
/* Servico */
*****/

package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Servico;
import controle.Conexao;

public class ServicoDAO {

    private Connection con;
    private Statement st = null;
    private ResultSet rs = null;

    // Inicializa a conexão no construtor
    public ServicoDAO() throws SQLException {
        con = Conexao.getConnection();
        st = con.createStatement();
    }

    //Método que exclui um registro da tabela Serviço do banco.
    public void remove(Servico servico) {

        try {

            // Monta a string sql
            String sql = "delete from servico where codigo = ";

```

```

        sql = sql + servico.getCodigo();
        int res = st.executeUpdate (sql);
        if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
                                                "Excluir",
JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException err) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
                                                + err, "Excluir",
JOptionPane.ERROR_MESSAGE);
        err.printStackTrace();
    }
}

//Método que consulta todos os registros da tabela Servico do banco.
public Servico[] retrieveAll() {

    Servico[] aux;
    Servico[] aux1 = new Servico[1];

    try {
        rs = st.executeQuery("select count(codigo) from servico");
        rs.next();
        int i = rs.getInt(1);

        rs = st.executeQuery("select * from servico order by codigo");
        aux = new Servico[i];

        while(rs.next())
        {
            int j= rs.getRow()-1;
            aux[j] = new Servico();
            aux[j].setCodigo(rs.getInt("codigo"));
            aux[j].setStatus(rs.getString("status"));
            aux[j].setHost(rs.getString("host"));
            aux[j].setPorta(rs.getInt("porta"));

        }
        return aux;

    } catch (SQLException err) {
        // Retorna uma mensagem informando o erro

        JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
dados!\nInformações sobre o erro:"
                                                + err, "Consultar",
JOptionPane.ERROR_MESSAGE);
        err.printStackTrace();
    }
    return aux1;
}

/*
public Servico retrieveByID(String nome) {
    return null;
}
*/

//Método que atualiza um registro da tabela Serviço do banco.
public void update(Servico servico) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql

```

```

String sql = "update servico set status = ?, host = ?, porta = ?";
sql = sql + " where codigo = ?";

// Passa a string para o PreparedStatement
pstm = conn.prepareStatement(sql);

// Coloca os verdadeiros valores no lugar dos ?
pstm.setString(1, servico.getStatus());
pstm.setString(2, servico.getHost());
pstm.setInt(3, servico.getPorta());
pstm.setInt(4, servico.getCodigo());
// Executa
pstm.execute();
} catch (SQLException err) {
// Retorna uma mensagem informando o erro
JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
dados!\nInformações sobre o erro:"
+ err, "Atualizar",
JOptionPane.ERROR_MESSAGE);
err.printStackTrace();
}
}

//Método que insere um registro na tabela Serviço do banco.
public void save(Servico servico) {

// Pega a conexão estabelecida
Connection conn = this.con;
// Cria um PreparedStatement
PreparedStatement pstm = null;

try {
// Monta a string sql
String sql = "insert into servico (status, host, porta)
values(?,?,?)";

// Passa a string para o PreparedStatement
pstm = conn.prepareStatement(sql);

// Coloca os verdadeiros valores no lugar dos ?
pstm.setString(1, servico.getStatus());
pstm.setString(2, servico.getHost());
pstm.setInt(3, servico.getPorta());
// Executa
pstm.execute();
} catch (SQLException err) {
// Retorna uma mensagem informando o erro
JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
+ err, "Salvar",
JOptionPane.ERROR_MESSAGE);
err.printStackTrace();
}
}
}

```

Classe *SistemaDAO.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */

```

```

/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe busca os dados no banco e atualiza a interface da Tabela */
/* Sistema */
/*****/

package DAO;

import java.sql.*;
import javax.swing.JOptionPane;
import modelo.Sistema;
import controle.Conexao;

public class SistemaDAO {

private Connection con;
private Statement st = null;
private ResultSet rs = null;

// Inicializa a conexão no construtor
public SistemaDAO() throws SQLException {
con = Conexao.getConnection();
st = con.createStatement();
}

//Método que exclui um registro da tabela Sistema do banco.
public void remove(Sistema sistema) {

try {
// Monta a string sql
String sql = "delete from sistema where Codigo = ";
sql = sql + sistema.getCodigo();
int res = st.executeUpdate (sql);
if (res > 0) JOptionPane.showMessageDialog(null, "Registro excluído
com sucesso!",
"Excluir",
JOptionPane.INFORMATION_MESSAGE);
} catch (SQLException e) {
// Retorna uma mensagem informando o erro
JOptionPane.showMessageDialog(null, "Não foi possível excluir o
registro!\nInformações sobre o erro:"
+ e, "Excluir",
JOptionPane.ERROR_MESSAGE);
e.printStackTrace();
}
}

//Método que consulta todos os registros da tabela Sistema do banco.
public Sistema[] retrieveAll() {

Sistema[] aux;
Sistema[] aux1 = new Sistema[1];

try {
rs = st.executeQuery("select count(Codigo) from sistema");
rs.next();
int i = rs.getInt(1);

rs = st.executeQuery("select * from sistema order by Codigo");
aux = new Sistema[i];

while(rs.next())
{
int j= rs.getRow()-1;
aux[j] = new Sistema();
aux[j].setCodigo(rs.getInt("Codigo"));
}
}
}
}

```

```

        aux[j].setNome(rs.getString("Nome"));
        aux[j].setStatus(rs.getString("Status"));
        aux[j].setDataInicio(rs.getDate("Data_inicio"));
        aux[j].setDataFim(rs.getDate("Data_fim"));
    }
    return aux;
} catch (SQLException e) {
    // Retorna uma mensagem informando o erro

    JOptionPane.showMessageDialog(null, "Não foi possível recuperar os
    dados!\nInformações sobre o erro:"
                                + e, "Consultar", JOptionPane.ERROR_MESSAGE);

    e.printStackTrace();
}
return aux1;
}
/*
public Sistema retrieveByID(String nome) {
    return null;
}
*/

//Método que atualiza um registro da tabela Sistema do banco.
public void update(Sistema sistema) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql
        String sql = "update sistema set Nome = ?, Status = ?, Data_inicio = ?,
        Data_fim = ?";
        sql = sql + " where Codigo = ?";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, sistema.getNome());
        pstmt.setString(2, sistema.getStatus());
        pstmt.setDate(3, sistema.getDataInicio());
        pstmt.setDate(4, sistema.getDataFim());
        pstmt.setInt(5, sistema.getCodigo());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível atualizar os
        dados!\nInformações sobre o erro:"
                                    + e, "Atualizar", JOptionPane.ERROR_MESSAGE);

        e.printStackTrace();
    }
}

//Método que insere um registro na tabela Sistema do banco.
public void save(Sistema sistema) {

    // Pega a conexão estabelecida
    Connection conn = this.con;
    // Cria um PreparedStatement
    PreparedStatement pstmt = null;

    try {
        // Monta a string sql

```

```

        String sql = "insert into sistema (Nome, Status, Data_inicio, Data_fim)
values(?, ?, ?, ?)";

        // Passa a string para o PreparedStatement
        pstmt = conn.prepareStatement(sql);

        // Coloca os verdadeiros valores no lugar dos ?
        pstmt.setString(1, sistema.getNome());
        pstmt.setString(2, sistema.getStatus());
        pstmt.setDate(3, sistema.getDataInicio());
        pstmt.setDate(4, sistema.getDataFim());
        // Executa
        pstmt.execute();
    } catch (SQLException e) {
        // Retorna uma mensagem informando o erro
        JOptionPane.showMessageDialog(null, "Não foi possível salvar os
dados!\nInformações sobre o erro:"
                                + e, "Salvar", JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
}

```

A.3 PACOTE INTERFACE_REDE

Classe *Cliente.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/*
/* Esta classe é responsável pela iniciativa da comunicação por parte do */
/* Gestor para fazer a monitoração e a calibração */
*****/
package interface_rede;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.net.InetAddress;
import javax.swing.JOptionPane;
import DAO.*;
import modelo.*;

public class Cliente {

//Envia um ping para o host nodo específico várias vezes até obter resposta,
//retornando o total destas tentativas até o máximo de 10.
public int getTentativas (String IPHost)
{
    int Tentativas = 1;
    boolean TentativaOk = false;
    while ((Tentativas < 10) && !TentativaOk)
    {
        try {
            if (InetAddress.getByName(IPHost).isReachable(2000))
                TentativaOk = true;

```

```

        else
            Tentativas++;
    }
    catch (Exception e) {
        Tentativas++;
    }
}
return Tentativas;
}

//Método que calcula o atraso de um nodo específico.
public double getAtraso (String IPhost, int portaHost)
{
    double tempoatual = (double)System.nanoTime();
    try {
        Socket socketcliente = new Socket(IPhost, portaHost);
        DataInputStream inSDU = new /*
DataInputStream(socketcliente.getInputStream());
        DataOutputStream outSDU = new
DataOutputStream(socketcliente.getOutputStream());
        Protocolo prot = new Protocolo();
        prot.setComandoUsuario("Enviar Ping");
        prot.executaIniciador();
        if (prot.getPrimitiva().equals("PINGrequest"))
        {
            outSDU.writeUTF(prot.getPDU());
            prot.setPDU(inSDU.readUTF());
            prot.setPrimitiva("PONGindication");
            prot.executaIniciador();
            if (prot.getComandoUsuario().equals("Get Atraso"))
            {
                tempoatual = (double)System.nanoTime() - tempoatual;
                tempoatual = tempoatual/Math.pow(10, 6);
            }
        }
        outSDU.close();
        socketcliente.close();
    }
    catch (Exception e) {
        return -1;
    }
}
return tempoatual;
}

//Verifica a quantidade de serviços de um nodo específico que estão com a porta
aberta.
public int getServicos (String IPhost, Servico[] servicos)
{
    int totserv = 0;
    System.out.println(IPhost);
    for (int i = 0; i < servicos.length; i++)
    {
        if (servicos[i].getStatus().equals("Ativo") &&
servicos[i].getHost().equals(IPhost))
        {
            System.out.println(i + " :porta " + servicos[i].getPorta() + "
"+servicos[i].getStatus());
            try
            {
                Socket ServerSok = new Socket(IPhost,servicos[i].getPorta());
                totserv++;
                //System.out.println(i + " :porta " + servicos[i].getPorta() + "
aberta");
                ServerSok.close();
            }
            catch (Exception e)

```



```

        { System.out.println(i + " :porta " + servicos[i].getPorta() + "
fechada"); }
    }
    }
    return totserv;
}

//Este método mede o atraso e a quantidade de serviços ativos para todos os nodos
que
//estão cadastrados como Fase no banco de dados.
public void executaCalibracao() {

    try {
        Servico[] servicos;
        ServicoDAO servicoDAO = new ServicoDAO();
        servicos = servicoDAO.retrieveAll();
        for (int i = 0; i < servicos.length; i++)
        {
            System.out.println(i+" "+servicos[i].getStatus());
            if (servicos[i].getStatus().equals("Fase"))
            {
                double atraso = getAtraso(servicos[i].getHost(),
servicos[i].getPorta());
                int totserv = getServicos(servicos[i].getHost(), servicos);
                Host h = new Host();
                h.setIP(servicos[i].getHost());
                h.setAtraso(atraso);
                h.setServicos(totserv);
                HostDAO hdao = new HostDAO();
                hdao.update(h);
            }
        }
        JOptionPane.showMessageDialog(null, "Calibração efetuada.",
            "Calibração", JOptionPane.INFORMATION_MESSAGE);
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}
}
}

```

Classe *Protocolo.java*

```

/*****
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável por executar o protocolo de rede. */
/*****

package interface_rede;

public class Protocolo {

    public static enum tipoEstado {REPOUSO, ESPERAART, ESPERAPONG};
    private tipoEstado estado = tipoEstado.REPOUSO;
    private String PDU = null;
    private String Primitiva = null;
    private String ComandoUsuario = null;

    public tipoEstado getEstado() {

```

```

        return estado;
    }

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

    public String getPDU() {
        return PDU;
    }

    public void setPDU(String pDU) {
        PDU = pDU;
    }

    public String getPrimitiva() {
        return Primitiva;
    }

    public void setPrimitiva(String primitiva) {
        Primitiva = primitiva;
    }

    public String getComandoUsuario() {
        return ComandoUsuario;
    }

    public void setComandoUsuario(String comandoUsuario) {
        ComandoUsuario = comandoUsuario;
    }

    //Este método é responsável pelo processamento do lado respondedor do protocolo.
    public void executaRespondedor() {

        switch (estado)
        {
            case REPOUSO:
                if (PDU.equals("ENVIAART"))
                    { estado = tipoEstado.ESPERAART; Primitiva = "RECEBEARTrequest";
                PDU = null;}
                break;
            case ESPERAART:
                if (Primitiva.equals("FIMARTindication"))
                    {estado = tipoEstado.REPOUSO; Primitiva = null; PDU = null;}
                break;
        }
    }

    //Este método é responsável pelo processamento do lado iniciador do protocolo.
    public void executaIniciador() {

        switch (estado)
        {
            case REPOUSO:
                if (ComandoUsuario.equals("Enviar Ping"))
                    { estado = tipoEstado.ESPERAPONG; Primitiva =
                "PINGrequest"; PDU = "PING";}
                break;
            case ESPERAPONG:
                if (Primitiva.equals("PONGindication") && PDU.equals("PONG"))
                    {estado = tipoEstado.REPOUSO; Primitiva = null; PDU = null;
                ComandoUsuario = "Get Atraso";}
                break;
        }
    }
}
Classe Servidor.java

```

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por aguardar a conexão de um nodo. */
/* *****/

package interface_rede;

import java.net.*;
import java.io.*;

public class Servidor extends Thread {

    int MinhaPorta;
    boolean listening = true;

    public Servidor (int p)
    {
        MinhaPorta = p;
    }

    public int getMinhaPorta() {
        return MinhaPorta;
    }

    public void setMinhaPorta(int minhaPorta) {
        MinhaPorta = minhaPorta;
    }

    public boolean isListening() {
        return listening;
    }

    public void setListening(boolean listening) {
        this.listening = listening;
    }

    //Método responsável por aguardar a conexão com o nodo, criando um processo
    servidor para
    //atender a cada nodo.
    public void iniciar () throws IOException {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(MinhaPorta);
        } catch (IOException e) {
            System.err.println("Não pude ler na porta: " + MinhaPorta);
            System.exit(-1);
        }

        while (listening)
            new ServidorThread(serverSocket.accept()).start();

        serverSocket.close();
    }

    public void run ()
    {
        try {
            iniciar();
        }
    }
}

```

```

    }
    catch (IOException e) {
        System.err.println("Não pude ler na porta: 4444."); }
}
}

```

Classe *ServidorThread.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável pelo processo servidor que atende a um nodo. */
/* *****/

package interface_rede;

import java.net.*;
import java.io.*;
import java.sql.SQLException;
import java.util.*;
import javax.swing.JOptionPane;
import DAO.*;
import modelo.*;

public class ServidorThread extends Thread {
    private Socket socket = null;
    private String IPhost;
    String nomeArqArt = "C:\\temp\\artefato_";

    public ServidorThread(Socket socket) {
        this.socket = socket;
    }

    //Método responsável por receber o arquivo de artefato enviado pelo nodo.
    public void recebeArtefato() {

        Date dataHora = new Date();
        nomeArqArt = nomeArqArt + dataHora.getTime() + ".pdf";

        try {
            DataOutputStream output = new DataOutputStream(new
            FileOutputStream(nomeArqArt));
            DataInputStream inputMsg = new DataInputStream(socket.getInputStream());

            int readByte = inputMsg.read();
            while (readByte !=-1) {
                output.write((byte)readByte);
                readByte = inputMsg.read();
            }
            inputMsg.close();
            output.close();
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

```

```

//Método principal que aciona o protocolo para dar início à recepção do arquivo do
artefato.
    public void run() {

        try {
            DataInputStream inSDU = new DataInputStream(socket.getInputStream());
            Protocolo prot = new Protocolo();

            IPhost = inSDU.readUTF();
            prot.setPDU(inSDU.readUTF());
            prot.executaRespondedor();
            if (prot.getPrimitiva().equals("RECEBEARTrequest"))
            {
                recebaArtefato();
                prot.setPrimitiva("FIMARTindication");
                prot.executaRespondedor();
            }
            Fase f = new Fase();
            FaseDAO faseDAO = new FaseDAO();
            f = faseDAO.retrieveByHost(IPhost);
            Atividade a = new Atividade();
            a.setHost(IPhost);System.out.println(IPhost);
            if (f.getDataFim().before(new Date()))
                { a.setStatus("Atrasado"); }
            else a.setStatus("Validando");
            a.setNomeArqArt(nomeArqArt);
            AtividadeDAO atividadeDAO = new AtividadeDAO();
            atividadeDAO.save(a);
            JOptionPane.showMessageDialog(null, "Artefato recebido do host: " +
IPhost,
                "Artefato", JOptionPane.INFORMATION_MESSAGE);
            inSDU.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
}

```

A.4 PACOTE INTERFACE_USUARIO

Classe *FormularioPrincipal.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/*
/* Esta classe processa a visualização da interface do Gestor de software na */
/* nuvem. */
/* *****/

package interface_usuario;

```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.BorderLayout;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import java.awt.Point;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JFrame;
import javax.swing.JDialog;
import java.awt.Dimension;
import javax.swing.JComboBox;
import java.awt.Rectangle;
import javax.swing.JTable;
import javax.swing.table.*;
import controle.*;
import javax.swing.JScrollPane;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JTextPane;

public class FormularioPrincipal {

    private JFrame jFrame = null;
    private JPanel jContentPane = null;
    private JMenuBar jMenuBar = null;
    private JMenu menuGerenciar = null;
    private JMenu menuMonitorar = null;
    private JMenu menuAjuda = null;
    private JMenuItem exitMenuItem = null;
    private JMenuItem aboutMenuItem = null;
    private JMenuItem iniciarMonitorMenuItem = null; // @jve:decl-index=0:
    private JMenuItem cancelarmonMenuItem = null;
    private JMenuItem calibrarMenuItem = null;
    private JDialog aboutDialog = null; // @jve:decl-index=0:visual-
constraint="941,47"
    private JPanel aboutContentPane = null;
    private JLabel aboutVersionLabel = null;
    private JFrame jAcessoBDFrame = null; // @jve:decl-index=0:visual-
constraint="424,2"
    private JPanel jAcessoBDpane = null;
    private JMenuItem manterBDMMenuItem = null; // @jve:decl-index=0:
    private JComboBox jComboTabelaBD = null;
    private JLabel jLabelTabelaBD = null;
    private GestorSoft gs = new GestorSoft(); // @jve:decl-index=0:
    private MonitorNodos mn = new MonitorNodos(); // @jve:decl-index=0:
    private JScrollPane jScrollPaneBD = null;
    private JTable jTableBD = null;
    private JButton jButtonInserir = null;
    private JButton jButtonExcluir = null;
    private JButton jButtonAtualizar = null;
    private JLabel jLabelMonitorando = null;
    private JLabel jLabelPorta = null;
    private JMenuItem mudarPortaMenuItem = null;
    private JTextPane jTextPaneMon = null;

    /**
     * This method initializes jAcessoBDFrame
     *
     * @return javax.swing.JFrame
     */
    private JFrame getJAcessoBDFrame() {
        if (jAcessoBDFrame == null) {

```

```

        jAcessoBDFrame = new JFrame();
        jAcessoBDFrame.setSize(new Dimension(627, 395));
        jAcessoBDFrame.setTitle("Manter Banco de Dados");
        jAcessoBDFrame.setMinimumSize(new Dimension(600, 300));
        jAcessoBDFrame.setPreferredSize(new Dimension(600, 400));
        jAcessoBDFrame.setContentPane(getJAcessoBDpane());
    }
    return jAcessoBDFrame;
}

/**
 * This method initializes jAcessoBDpane
 *
 * @return javax.swing.JPanel
 */
private JPanel getJAcessoBDpane() {
    if (jAcessoBDpane == null) {
        jLabelTabelaBD = new JLabel();
        jLabelTabelaBD.setBounds(new Rectangle(58, 12, 191, 24));
        jLabelTabelaBD.setText("Escolha a Tabela a Ser Mantida:");
        jAcessoBDpane = new JPanel();
        jAcessoBDpane.setLayout(null);
        jAcessoBDpane.add(getJComboTabelaBD(), null);
        jAcessoBDpane.add(jLabelTabelaBD, null);
        jAcessoBDpane.add(getJScrollPaneBD(), null);
        jAcessoBDpane.add(getJButtonInserir(), null);
        jAcessoBDpane.add(getJButtonExcluir(), null);
        jAcessoBDpane.add(getJButtonAtualizar(), null);
    }
    return jAcessoBDpane;
}

/**
 * This method initializes manterBDMMenuItem
 *
 * @return javax.swing.JMenuItem
 */
private JMenuItem getManterBDMMenuItem() {
    if (manterBDMMenuItem == null) {
        manterBDMMenuItem = new JMenuItem();
        manterBDMMenuItem.setText("Manter Banco de Dados");
        manterBDMMenuItem.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e)
{
        JFrame manterBDframe = getJAcessoBDFrame();
        manterBDframe.pack();
        Point loc = getJFrame().getLocation();
        loc.translate(20, 20);
        manterBDframe.setLocation(loc);
        jComboTabelaBD.setSelectedIndex(-1);

        TableModel modelo = new DefaultTableModel(0,0);
        jTableBD.setModel(modelo);
        jTableBD.setPreferredSize(new Dimension(510, 200));

jTableBD.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);

        manterBDframe.setVisible(true);
    }
});
    }
    return manterBDMMenuItem;
}

/**
 * This method initializes jComboTabelaBD
 *

```

```

    * @return javax.swing.JComboBox
    */
private JComboBox getJComboTabelaBD() {
    String[] arrayTabelas = {"Sistema", "Equipe", "Fase", "Atividade",
        "Host", "Serviço", "Monitoramento"};
    if (jComboTabelaBD == null) {
        jComboTabelaBD = new JComboBox(arrayTabelas);
        jComboTabelaBD.setBounds(new Rectangle(249, 12, 298, 25));
        jComboTabelaBD.setSelectedIndex(-1);
        jComboTabelaBD.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                switch (jComboTabelaBD.getSelectedIndex()) {
                    case 0: gs.consultarSistema(getJTableBD());
break;
                    case 1: gs.consultarEquipe(getJTableBD());
break;
                    case 2: gs.consultarFase(getJTableBD()); break;
                    case 3: gs.consultarAtividade(getJTableBD());
break;
                    case 4: gs.consultarHost(getJTableBD()); break;
                    case 5: gs.consultarServico(getJTableBD());
break;
                    case 6:
gs.consultarMonitoramento(getJTableBD()); break;
                }
            }
        });
    }
    return jComboTabelaBD;
}

/**
 * This method initializes jFrame
 *
 * @return javax.swing.JFrame
 */
private JFrame getJFrame() {
    if (jFrame == null) {
        jFrame = new JFrame();
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jFrame.setJMenuBar(getJJMenuBar());
        jFrame.setSize(392, 286);
        jFrame.setContentPane(getJContentPane());
        jFrame.setTitle("Gestor de Software na Nuvem");
        Thread thread = new MonitorNodosThread(jLabelMonitorando,
jTextPaneMon, mn);
        thread.start();
    }
    return jFrame;
}

/**
 * This method initializes jContentPane
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane() {
    if (jContentPane == null) {
        jLabelPorta = new JLabel();
        jLabelPorta.setBounds(new Rectangle(14, 189, 188, 16));
        jLabelPorta.setText("VALOR DA MINHA PORTA: nulo");
        jLabelMonitorando = new JLabel();
        jLabelMonitorando.setBounds(new Rectangle(15, 13, 349, 21));
        jLabelMonitorando.setForeground(Color.blue);
        jLabelMonitorando.setFont(new Font("Dialog", Font.BOLD, 14));
        jLabelMonitorando.setText("");
    }
}

```



```

        jLabelMonitorando.setVisible(false);
        jContentPane = new JPanel();
        jContentPane.setLayout(null);
        jContentPane.add(jLabelMonitorando, null);
        jContentPane.add(jLabelPorta, null);
        jContentPane.add(getJTextPaneMon(), null);
    }
    return jContentPane;
}

/**
 * This method initializes jJMenuBar
 *
 * @return javax.swing.JMenuBar
 */
private JMenuBar getJJMenuBar() {
    if (jJMenuBar == null) {
        jJMenuBar = new JMenuBar();
        jJMenuBar.add(getMenuGerenciar());
        jJMenuBar.add(getMenuMonitorar());
        jJMenuBar.add(getMenuAjuda());
    }
    return jJMenuBar;
}

/**
 * This method initializes jMenu
 *
 * @return javax.swing.JMenu
 */
private JMenu getMenuGerenciar() {
    if (menuGerenciar == null) {
        menuGerenciar = new JMenu();
        menuGerenciar.setText("Gerenciar");
        menuGerenciar.add(getMudarPortaMenuItem());
        menuGerenciar.add(getCalibrarMenuItem());
        menuGerenciar.add(getManterBDMMenuItem());
        menuGerenciar.add(getExitMenuItem());
    }
    return menuGerenciar;
}

/**
 * This method initializes jMenu
 *
 * @return javax.swing.JMenu
 */
private JMenu getMenuMonitorar() {
    if (menuMonitorar == null) {
        menuMonitorar = new JMenu();
        menuMonitorar.setText("Monitorar");
        menuMonitorar.setActionCommand("Monitorar");
        menuMonitorar.add(getIniciarMonitorMenuItem());
        menuMonitorar.add(getCancelarmonMenuItem());
    }
    return menuMonitorar;
}

/**
 * This method initializes jMenu
 *
 * @return javax.swing.JMenu
 */
private JMenu getMenuAjuda() {
    if (menuAjuda == null) {
        menuAjuda = new JMenu();
        menuAjuda.setText("Ajuda");
        menuAjuda.add(getAboutMenuItem());
    }
}

```

```

        }
        return menuAjuda;
    }

    /**
     * This method initializes jMenuItem
     *
     * @return javax.swing.JMenuItem
     */
    private JMenuItem getExitMenuItem() {
        if (exitMenuItem == null) {
            exitMenuItem = new JMenuItem();
            exitMenuItem.setText("Sair");
            exitMenuItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }
            });
        }
        return exitMenuItem;
    }

    /**
     * This method initializes jMenuItem
     *
     * @return javax.swing.JMenuItem
     */
    private JMenuItem getAboutMenuItem() {
        if (aboutMenuItem == null) {
            aboutMenuItem = new JMenuItem();
            aboutMenuItem.setText("Sobre");
            aboutMenuItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    JDialog aboutDialog = getAboutDialog();
                    aboutDialog.pack();
                    Point loc = getJFrame().getLocation();
                    loc.translate(20, 20);
                    aboutDialog.setLocation(loc);
                    aboutDialog.setVisible(true);
                }
            });
        }
        return aboutMenuItem;
    }

    /**
     * This method initializes aboutDialog
     *
     * @return javax.swing.JDialog
     */
    private JDialog getAboutDialog() {
        if (aboutDialog == null) {
            aboutDialog = new JDialog(getJFrame(), true);
            aboutDialog.setTitle("Sobre");
            aboutDialog.setSize(new Dimension(275, 96));
            aboutDialog.setMinimumSize(new Dimension(300, 100));
            aboutDialog.setContentPane(getAboutContentPane());
        }
        return aboutDialog;
    }

    /**
     * This method initializes aboutContentPane
     *
     * @return javax.swing.JPanel
     */
    private JPanel getAboutContentPane() {
        if (aboutContentPane == null) {

```

```

        aboutContentPane = new JPanel();
        aboutContentPane.setLayout(new BorderLayout());
        aboutContentPane.add(getAboutVersionLabel(),
BorderLayout.CENTER);
    }
    return aboutContentPane;
}

/**
 * This method initializes aboutVersionLabel
 *
 * @return javax.swing.JLabel
 */
private JLabel getAboutVersionLabel() {
    if (aboutVersionLabel == null) {
        aboutVersionLabel = new JLabel();
        aboutVersionLabel.setText("Gestor de Software na Nuvem - Versão
1.0");
        aboutVersionLabel.setHorizontalAlignment(SwingConstants.CENTER);
    }
    return aboutVersionLabel;
}

/**
 * This method initializes jMenuItem
 *
 * @return javax.swing.JMenuItem
 */
private JMenuItem getIniciarMonitorMenuItem() {
    if (iniciarMonitorMenuItem == null) {
        iniciarMonitorMenuItem = new JMenuItem();
        iniciarMonitorMenuItem.setText("Iniciar Monitoramento");

        iniciarMonitorMenuItem.setActionCommand("Iniciar_Monitoramento");
        iniciarMonitorMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                mn.setPararMonitoramento(false);
                mn.setAcheiProbSeg(false);
                jLabelMonitorando.setVisible(true);
                jTextPaneMon.setVisible(true);
            }
        });
    }
    return iniciarMonitorMenuItem;
}

/**
 * This method initializes jMenuItem
 *
 * @return javax.swing.JMenuItem
 */
private JMenuItem getCancelarmonMenuItem() {
    if (cancelarmonMenuItem == null) {
        cancelarmonMenuItem = new JMenuItem();
        cancelarmonMenuItem.setText("Cancelar Monitoramento");
        cancelarmonMenuItem.setActionCommand("CancelarMonitoramento");
        cancelarmonMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                mn.setPararMonitoramento(true);
                try {
                    JOptionPane.showMessageDialog
                    (null, "Aguarde conclusão do
monitoramento",

```

```

        "Monitoramento",
JOptionPane.INFORMATION_MESSAGE);
        Thread.sleep(5000);
    } catch (Exception ez) {}
    //jLabelMonitorando.setVisible(false);
    //jTextPaneMon.setVisible(false);
    //jLabelMonitorando.setText(" ");
    //jTextPaneMon.setText(" ");
    }
    });
    }
    return cancelarmonMenuItem;
}

/**
 * This method initializes jMenuItem
 *
 * @return javax.swing.JMenuItem
 */
private JMenuItem getCalibrarMenuItem() {
    if (calibrarMenuItem == null) {
        calibrarMenuItem = new JMenuItem();
        calibrarMenuItem.setText("Calibrar");
        calibrarMenuItem.setActionCommand("Calibrar");
        calibrarMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                gs.calibrar();
            }
        });
    }
    return calibrarMenuItem;
}

/**
 * This method initializes jScrollPaneBD
 *
 * @return javax.swing.JScrollPane
 */
private JScrollPane getJScrollPaneBD() {
    if (jScrollPaneBD == null) {
        jScrollPaneBD = new JScrollPane();
        jScrollPaneBD.setPreferredSize(new Dimension(453, 200));
        jScrollPaneBD.setLocation(new Point(32, 60));
        jScrollPaneBD.setSize(new Dimension(528, 200));
        jScrollPaneBD.setBorder(null);

        jScrollPaneBD.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_A
S_NEEDED);

        jScrollPaneBD.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NE
EDED);

        jScrollPaneBD.setViewportViewView(getJTableBD());
    }
    return jScrollPaneBD;
}

/**
 * This method initializes jTableBD
 *
 * @return javax.swing.JTable
 */
private JTable getJTableBD() {
    if (jTableBD == null) {
        jTableBD = new JTable();
        jTableBD.setPreferredSize(new Dimension(510, 170));
        jTableBD.setLocation(new Point(0, 0));
    }
}

```

```

        jTableBD.setSize(new Dimension(510, 200));

jTableBD.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
    }
    return jTableBD;
}

/**
 * This method initializes jButtonInserir
 *
 * @return javax.swing.JButton
 */
private JButton getJButtonInserir() {
    if (jButtonInserir == null) {
        jButtonInserir = new JButton();
        jButtonInserir.setText("Inserir Nova Linha");
        jButtonInserir.setSize(new Dimension(180, 30));
        jButtonInserir.setLocation(new Point(15, 286));
        jButtonInserir.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                switch (jComboTabelaBD.getSelectedIndex()) {
break;
                    case 0: gs.inserirLinhaSistema(getJTableBD());
break;
                    case 1: gs.inserirLinhaEquipe(getJTableBD());
break;
                    case 2: gs.inserirLinhaFase(getJTableBD());
break;
                    case 3: gs.inserirLinhaAtividade(getJTableBD());
break;
                    case 4: gs.inserirLinhaHost(getJTableBD());
break;
                    case 5: gs.inserirLinhaServico(getJTableBD());
break;
                    case 6:
gs.inserirLinhaMonitoramento(getJTableBD()); break;
                }
            }
        });
    }
    return jButtonInserir;
}

/**
 * This method initializes jButtonExcluir
 *
 * @return javax.swing.JButton
 */
private JButton getJButtonExcluir() {
    if (jButtonExcluir == null) {
        jButtonExcluir = new JButton();
        jButtonExcluir.setText("Excluir Registro");
        jButtonExcluir.setSize(new Dimension(180, 30));
        jButtonExcluir.setLocation(new Point(205, 286));
        jButtonExcluir.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                switch (jComboTabelaBD.getSelectedIndex()) {
                    case 0: gs.excluirSistema(getJTableBD()); break;
                    case 1: gs.excluirEquipe(getJTableBD()); break;
                    case 2: gs.excluirFase(getJTableBD()); break;
                    case 3: gs.excluirAtividade(getJTableBD());
break;
                    case 4: gs.excluirHost(getJTableBD()); break;
                    case 5: gs.excluirServico(getJTableBD()); break;

```

```

                                case 6: gs.excluirMonitoramento(getJTableBD());
break;
                                }
                                }
                                });
                                }
                                return jButtonExcluir;
                                }

/**
 * This method initializes jButtonAtualizar
 *
 * @return javax.swing.JButton
 */
private JButton getJButtonAtualizar() {
    if (jButtonAtualizar == null) {
        jButtonAtualizar = new JButton();
        jButtonAtualizar.setLocation(new Point(397, 286));
        jButtonAtualizar.setText("Atualizar Banco de Dados");
        jButtonAtualizar.setSize(new Dimension(180, 30));
        jButtonAtualizar.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
            {
                switch (jComboTabelaBD.getSelectedIndex()) {
                    case 0: gs.atualizarBDSistema(getJTableBD());
break;
                    case 1: gs.atualizarBDEquipe(getJTableBD());
break;
                    case 2: gs.atualizarBDFase(getJTableBD());
break;
                    case 3: gs.atualizarBDAtividade(getJTableBD());
break;
                    case 4: gs.atualizarBDHost(getJTableBD());
break;
                    case 5: gs.atualizarBDServico(getJTableBD());
break;
                    case 6:
gs.atualizarBDMonitoramento(getJTableBD()); break;
                }
            }
        });
    }
    return jButtonAtualizar;
}

/**
 * This method initializes mudarPortaMenuItem
 *
 * @return javax.swing.JMenuItem
 */
private JMenuItem getMudarPortaMenuItem() {
    if (mudarPortaMenuItem == null) {
        mudarPortaMenuItem = new JMenuItem();
        mudarPortaMenuItem.setText("Definir Minha Porta");
        mudarPortaMenuItem.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
            {
                gs.alterarPorta(jLabelPorta);
            }
        });
    }
    return mudarPortaMenuItem;
}

/**
 * This method initializes jTextPaneMon

```

```

*
* @return javax.swing.JTextPane
*/
private JTextPane getJTextPaneMon() {
    if (jTextPaneMon == null) {
        jTextPaneMon = new JTextPane();
        jTextPaneMon.setBounds(new Rectangle(16, 46, 347, 135));
        jTextPaneMon.setEditable(false);
        jTextPaneMon.setText("");
        jTextPaneMon.setVisible(false);
    }
    return jTextPaneMon;
}

/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            FormularioPrincipal application = new
FormularioPrincipal();
            application.getJFrame().setVisible(true);
        }
    });
}
}

```

Classe *InterfaceManterAtividade.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Atividade */
/* *****/

package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Atividade;

public class InterfaceManterAtividade {

    //Preenche a tabela da interface do formulário principal com os dados da tabela
    Atividade do banco.
    public void preencheTabelaAtividade(Atividade[] a, JTable jTable)
    {
        TableModel modelo = new DefaultTableModel(a.length,4){
            public boolean isCellEditable(int rowIndex, int mColIndex) {
                if (mColIndex == 2) return true; else return false;
            }
        };
    }
};

```

```

jTable.setModel(modelo);
jTable.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
jTable.setPreferredSize(new Dimension(500, 170+16*a.length));
jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
jTable.getColumnModel().getColumn(0).setHeaderValue("Código");
jTable.getColumnModel().getColumn(1).setHeaderValue("Host");
jTable.getColumnModel().getColumn(2).setHeaderValue("Status");
jTable.getColumnModel().getColumn(3).setHeaderValue("Arquivo do
Artefato");

for(int i = 0;i<a.length;i++)
{
    jTable.setValueAt(a[i].getCodigo(), i, 0);
    jTable.setValueAt(a[i].getHost(), i, 1);
    jTable.setValueAt(a[i].getStatus(), i, 2);
    jTable.setValueAt(a[i].getNomeArqArt(), i, 3);
}

}

//Este método retorna a quantidade de registros já existentes na tabela Atividade
mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int qra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0")) qra++;
    }
    return qra;
}

//Este método preenche um vetor de objetos atividade com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Atividade[] a, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0"))
        {
            a[j].setCodigo(Integer.parseInt(jTable.getValueAt(i,
0).toString()));
            a[j].setHost(jTable.getValueAt(i, 1).toString());
            a[j].setStatus(jTable.getValueAt(i, 2).toString());
            a[j].setNomeArqArt(jTable.getValueAt(i, 3).toString());
            j++;
        }
    }
}

//Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela
da interface
//do formulário principal.
public boolean existeLinhaSelecionada (JTable jTable)
{
    if (jTable.getSelectedRow() == -1) return false;
    return true;
}
}

```


Classe *InterfaceManterEquipe.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Equipe */
/* *****/

package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Equipe;

public class InterfaceManterEquipe {

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
public void preencheTabelaEquipe(Equipe[] e, JTable jTable)
{
    TableModel modelo = new DefaultTableModel(e.length,4){
        public boolean isCellEditable(int rowIndex, int mColIndex) {
            if (mColIndex == 0) return false; else return true;
        }
    };

    jTable.setModel(modelo);
    jTable.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
    jTable.setPreferredSize(new Dimension(510, 170+16*e.length));
    jTable.setSelectionModel(ListSelectionModel.SINGLE_SELECTION);
    jTable.getColumnModel().getColumn(0).setHeaderValue("Código do
Funcionário");
    jTable.getColumnModel().getColumn(1).setHeaderValue("Nome do
Funcionário");
    jTable.getColumnModel().getColumn(2).setHeaderValue("Cargo do
Funcionário");
    jTable.getColumnModel().getColumn(3).setHeaderValue("Código da Fase");

    for(int i = 0;i<e.length;i++)
    {
        jTable.setValueAt(e[i].getCodigo_funcionario(), i, 0);
        jTable.setValueAt(e[i].getNome_funcionario(), i, 1);
        jTable.setValueAt(e[i].getCargo_funcionario(), i, 2);
        jTable.setValueAt(e[i].getCodigo_fase(), i, 3);
    }
}

//Método responsável por criar uma linha nova em branco na tabela da interface do
formulário
//principal.
public void criaLinhaTabelaEquipe (JTable jTable)

```

```

{
    DefaultTableModel model = (DefaultTableModel) jTable.getModel();
    model.addRow(new Object[]{"0", "", "", ""});
}

//Este método retorna a quantidade de registros novos na tabela Equipe mostrados na
//interface do formulário.
public int quantRegistrosNovos (JTable jTable)
{
    int qrn = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0")) qrn++;
    }
    return qrn;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros novos.
public void preencheRegistrosNovos (Equipe[] e, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0"))
        {
            e[j].setNome_funcionario(jTable.getValueAt(i, 1).toString());
            e[j].setCargo_funcionario(jTable.getValueAt(i, 2).toString());
            e[j].setCodigo_fase(Integer.parseInt(jTable.getValueAt(i,
3).toString()));
            j++;
        }
    }
}

//Este método retorna a quantidade de registros já existentes na tabela Atividade
mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int qra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0")) qra++;
    }
    return qra;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Equipe[] e, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0"))
        {
            e[j].setCodigo_funcionario(Integer.parseInt(jTable.getValueAt(i,
0).toString()));
            e[j].setNome_funcionario(jTable.getValueAt(i, 1).toString());
            e[j].setCargo_funcionario(jTable.getValueAt(i, 2).toString());
            e[j].setCodigo_fase(Integer.parseInt(jTable.getValueAt(i,
3).toString()));
        }
    }
}

```

```

                j++;
            }
        }
    }

    //Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela
    da interface
    //do formulário principal.
    public boolean existeLinhaSelecionada (JTable jTable)
    {
        if (jTable.getSelectedRow() == -1) return false;
        return true;
    }

    //Este método preenche um objeto equipe com os dados da linha selecionada na tabela
    da interface
    //do formulário principal.
    public void getRegistroSelecionado (Equipe e, JTable jTable)
    {
        int l = jTable.getSelectedRow();
        e.setCodigo_funcionario (Integer.parseInt(jTable.getValueAt(l,
0).toString()));
        e.setNome_funcionario(jTable.getValueAt(l, 1).toString());
        e.setCargo_funcionario(jTable.getValueAt(l, 2).toString());
        e.setCodigo_fase(Integer.parseInt(jTable.getValueAt(l, 3).toString()));
    }
}

```

Classe *InterfaceManterFase.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Fase */
/* *****/
package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Fase;

public class InterfaceManterFase {

    //Este método preenche um vetor de objetos fase com todas as informações alteradas
    ou não
    //da tabela da interface do formulário dos registros.
    public void preencheTabelaFase (Fase[] f, JTable jTable)
    {
        TableModel modelo = new DefaultTableModel(f.length,7){
            public boolean isCellEditable(int rowIndex, int mColIndex) {
                if (mColIndex == 0) return false; else return true;
            }
        };
    };
}

```

```

jTable.setModel(modelo);
jTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
jTable.setPreferredSize(new Dimension(1000, 170+16*f.length));
    jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    jTable.getColumnModel().getColumn(0).setHeaderValue("Código");
    jTable.getColumnModel().getColumn(1).setHeaderValue("Nome");
    jTable.getColumnModel().getColumn(2).setHeaderValue("Host");
    jTable.getColumnModel().getColumn(3).setHeaderValue("Status");
    jTable.getColumnModel().getColumn(4).setHeaderValue("Data Inicial");
    jTable.getColumnModel().getColumn(5).setHeaderValue("Data Final");

    jTable.getColumnModel().getColumn(6).setHeaderValue("Código do
sistema");

    for(int i = 0;i<f.length;i++)
    {
        jTable.setValueAt(f[i].getCodigo(), i, 0);
        jTable.setValueAt(f[i].getNome(), i, 1);
        jTable.setValueAt(f[i].getHost(), i, 2);
        jTable.setValueAt(f[i].getStatus(), i, 3);
        jTable.setValueAt(f[i].getDataInicio(), i, 4);
        jTable.setValueAt(f[i].getDataFim(), i, 5);
        jTable.setValueAt(f[i].getCodigoSistema(), i, 6);
    }
}

//Método responsável por criar uma linha nova em branco na tabela da interface do
formulário
//principal.
public void criaLinhaTabelaFase (JTable jTable)
{
    DefaultTableModel model = (DefaultTableModel) jTable.getModel();
    model.addRow(new Object[]{"0", "", "", "", "", "", ""});
}

//Este método retorna a quantidade de registros novos na tabela Fase mostrados na
//interface do formulário.
public int quantRegistrosNovos (JTable jTable)
{
    int qrn = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0")) qrn++;
    }
    return qrn;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros novos.
public void preencheRegistrosNovos (Fase[] f, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0"))
        {
            f[j].setNome(jTable.getValueAt(i, 1).toString());
            f[j].setHost(jTable.getValueAt(i, 2).toString());
            f[j].setStatus(jTable.getValueAt(i, 3).toString());
            f[j].setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(i,
4).toString()));
            f[j].setDataFim(java.sql.Date.valueOf(jTable.getValueAt(i,
5).toString()));

```

```

        f[j].setCodigoSistema(Integer.parseInt(jTable.getValueAt(i,
6).toString()));
        j++;
    }
}

//Este método retorna a quantidade de registros já existentes na tabela Atividade
mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int gra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0")) gra++;
    }
    return gra;
}

//Este método preenche um vetor de objetos fase com todas as informações alteradas
ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Fase[] f, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0"))
        {
            f[j].setCodigo(Integer.parseInt(jTable.getValueAt(i,
0).toString()));
            f[j].setNome(jTable.getValueAt(i, 1).toString());
            f[j].setHost(jTable.getValueAt(i, 2).toString());
            f[j].setStatus(jTable.getValueAt(i, 3).toString());
            f[j].setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(i,
4).toString()));
            f[j].setDataFim(java.sql.Date.valueOf(jTable.getValueAt(i,
5).toString()));
            f[j].setCodigoSistema(Integer.parseInt(jTable.getValueAt(i,
6).toString()));
            j++;
        }
    }
}

//Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela
da interface
//do formulário principal.
public boolean existeLinhaSelecionada (JTable jTable)
{
    if (jTable.getSelectedRow() == -1) return false;
    return true;
}

//Este método preenche um objeto fase com os dados da linha selecionada na tabela
da interface
//do formulário principal.
public void getRegistroSelecionado (Fase f, JTable jTable)
{
    int l = jTable.getSelectedRow();
    f.setCodigo(Integer.parseInt(jTable.getValueAt(l, 0).toString()));

    f.setNome(jTable.getValueAt(l, 1).toString());
    f.setHost(jTable.getValueAt(l, 2).toString());
    f.setStatus(jTable.getValueAt(l, 3).toString());
}

```

```

        f.setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(1, 4).toString()));
        f.setDataFim(java.sql.Date.valueOf(jTable.getValueAt(1, 5).toString()));

        f.setCodigoSistema(Integer.parseInt(jTable.getValueAt(1, 6).toString()));
    }
}

```

Classe *InterfaceManterHost.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Host */
/* *****/

package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Host;

public class InterfaceManterHost {

    //Este método preenche um vetor de objetos host com todas as informações alteradas
    ou não
    //da tabela da interface do formulário dos registros.
    public void preencheTabelaHost(Host[] h, JTable jTable)
    {
        TableModel modelo = new DefaultTableModel(h.length,3){
            public boolean isCellEditable(int rowIndex, int mColIndex) {
                if (mColIndex == 0) return true; else return false;
            }
        };

        jTable.setModel(modelo);
        jTable.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
        jTable.setPreferredSize(new Dimension(510, 170+16*h.length));
        jTable.setSelectionModel(ListSelectionModel.SINGLE_SELECTION);
        jTable.getColumnModel().getColumn(0).setHeaderValue("Endereço
IP");
        jTable.getColumnModel().getColumn(1).setHeaderValue("Total de
Serviços");
        jTable.getColumnModel().getColumn(2).setHeaderValue("Valor do
Atraso");

        for(int i = 0; i < h.length; i++)
        {
            jTable.setValueAt(h[i].getIP(), i, 0);
            jTable.setValueAt(h[i].getServicos(), i, 1);
            jTable.setValueAt(h[i].getAtraso(), i, 2);
        }
    }
}

```

```

//Método responsável por criar uma linha nova em branco na tabela da
interface do formulário
//principal.
public void criaLinhaTabelaHost (JTable jTable)
{
    DefaultTableModel model = (DefaultTableModel) jTable.getModel();
    model.addRow(new Object[]{"0.0.0.0", "-1", "0"});
}

//Este método retorna a quantidade de novos registros existentes na tabela
Host mostrados na
//interface do formulário.
public int quantRegistrosNovos (JTable jTable)
{
    int qrn = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 1).equals("-1")) qrn++;
    }
    return qrn;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
public void preencheRegistrosNovos (Host[] h, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 1).equals("-1"))
        {
            h[j].setIP(jTable.getValueAt(i, 0).toString());
            h[j].setServicos(Integer.parseInt(jTable.getValueAt(i,
1).toString()));
            h[j].setAtraso(Double.parseDouble(jTable.getValueAt(i,
2).toString()));
            j++;
        }
    }
}

//Este método retorna a quantidade de registros já existentes na tabela host
mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int qra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 1).equals("-1")) qra++;
    }
    return qra;
}

//Este método preenche um vetor de objetos atividade com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Host[] h, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 1).equals("-1"))

```

```

        {
            h[j].setIP(jTable.getValueAt(i, 0).toString());
            h[j].setServicos(Integer.parseInt(jTable.getValueAt(i,
1).toString()));
            h[j].setAtraso(Double.parseDouble(jTable.getValueAt(i,
2).toString()));
            j++;
        }
    }

    //Este método apenas retorna verdadeiro se existe uma linha selecionada na
tabela da interface
    //do formulário principal.
    public boolean existeLinhaSelecionada (JTable jTable)
    {
        if (jTable.getSelectedRow() == -1) return false;
        return true;
    }

    //Este método preenche um objeto host com os dados da linha selecionada na
tabela da interface
    //do formulário principal.
    public void getRegistroSelecionado (Host h, JTable jTable)
    {
        int l = jTable.getSelectedRow();
        h.setIP (jTable.getValueAt(l, 0).toString());
        h.setServicos(Integer.parseInt(jTable.getValueAt(l, 1).toString()));
        h.setAtraso(Double.parseDouble(jTable.getValueAt(l, 2).toString()));
    }
}

```

Classe *InterfaceManterMonitoramento.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Monitoramento */
/* *****/
package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Monitoramento;

public class InterfaceManterMonitoramento {

    //Este método preenche um vetor de objetos monitoramento com todas as informações
alteradas ou não
    //da tabela da interface do formulário dos registros.
    public void preencheTabelaMonitoramento(Monitoramento[] m, JTable jTable)
    {
        TableModel modelo = new DefaultTableModel(m.length,9){
            public boolean isCellEditable(int rowIndex, int mColIndex) {

```



```

        return false;
    }
};

jTable.setModel(modelo);
jTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
jTable.setPreferredSize(new Dimension(2000, 170+16*m.length));
    jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    jTable.getColumnModel().getColumn(0).setHeaderValue("Código");
    jTable.getColumnModel().getColumn(1).setHeaderValue("Host");
    jTable.getColumnModel().getColumn(2).setHeaderValue("Atraso");
    jTable.getColumnModel().getColumn(3).setHeaderValue("Serviços");

    jTable.getColumnModel().getColumn(4).setHeaderValue("Tentativas");
    jTable.getColumnModel().getColumn(5).setHeaderValue("Índice de
Problema na Rede");
    jTable.getColumnModel().getColumn(6).setHeaderValue("Artefatos
Errados");
    jTable.getColumnModel().getColumn(7).setHeaderValue("Artefatos
Atrasados");
    jTable.getColumnModel().getColumn(8).setHeaderValue("Índice de
Problema de Segurança");

    for(int i = 0;i<m.length;i++)
    {
        jTable.setValueAt(m[i].getCodigo(), i, 0);
        jTable.setValueAt(m[i].getHost(), i, 1);
        jTable.setValueAt(m[i].getAtraso(), i, 2);
        jTable.setValueAt(m[i].getServicos(), i, 3);

        jTable.setValueAt(m[i].getTentativas(), i, 4);
        jTable.setValueAt(m[i].getIndiceProbRede(), i, 5);
        jTable.setValueAt(m[i].getArtefatosErrados(), i, 6);
        jTable.setValueAt(m[i].getArtefatosAtrasados(), i, 7);

        jTable.setValueAt(m[i].getIndiceProbSeg(), i, 8);
    }
}

//Este método apenas retorna verdadeiro se existe uma linha selecionada na
tabela da interface
//do formulário principal.
public boolean existeLinhaSelecionada (JTable jTable)
{
    if (jTable.getSelectedRow() == -1) return false;
    return true;
}

//Este método preenche um objeto monitoracao com os dados da linha
selecionada na tabela da interface
//do formulário principal.
public void getRegistroSelecionado (Monitoramento m, JTable jTable)
{
    int l = jTable.getSelectedRow();
    m.setCodigo (Integer.parseInt(jTable.getValueAt(l, 0).toString()));
    m.setHost(jTable.getValueAt(l, 1).toString());
    m.setAtraso(Double.parseDouble(jTable.getValueAt(l, 2).toString()));
    m.setServicos(Integer.parseInt(jTable.getValueAt(l, 3).toString()));
    m.setTentativas(Integer.parseInt(jTable.getValueAt(l, 4).toString()));
    m.setIndiceProbRede(Double.parseDouble(jTable.getValueAt(l,
5).toString()));
    m.setArtefatosErrados(Integer.parseInt(jTable.getValueAt(l,
6).toString()));
    m.setArtefatosAtrasados(Integer.parseInt(jTable.getValueAt(l,
7).toString()));
    m.setIndiceProbSeg(Double.parseDouble(jTable.getValueAt(l,
8).toString()));
}

```

```

    }
}

```

Classe *InterfaceManterServico.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* */
/* Esta classe é responsável por manter a interface da tabela Servico */
/* *****/

package interface_usuario;

import java.awt.Dimension;

import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Servico;

public class InterfaceManterServico {

//Este método preenche um vetor de objetos serviço com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
    public void preencheTabelaServico(Servico[] serv, JTable jTable)
    {
        TableModel modelo = new DefaultTableModel(serv.length,4){
            public boolean isCellEditable(int rowIndex, int mColIndex) {
                if (mColIndex == 0) return false; else return true;
            }
        };

        jTable.setModel(modelo);
        jTable.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
        jTable.setPreferredSize(new Dimension(510, 170+16*serv.length));
        jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        jTable.getColumnModel().getColumn(0).setHeaderValue("Código");
        jTable.getColumnModel().getColumn(1).setHeaderValue("Status");
        jTable.getColumnModel().getColumn(2).setHeaderValue("Host");
        jTable.getColumnModel().getColumn(3).setHeaderValue("Porta");

        for(int i = 0;i<serv.length;i++)
        {
            jTable.setValueAt(serv[i].getCodigo(), i, 0);
            jTable.setValueAt(serv[i].getStatus(), i, 1);
            jTable.setValueAt(serv[i].getHost(), i, 2);
            jTable.setValueAt(serv[i].getPorta(), i, 3);
        }
    }

//Método responsável por criar uma linha nova em branco na tabela da interface do
formulário

```

```

//principal.
public void criaLinhaTabelaServico (JTable jTable)
{
    DefaultTableModel model = (DefaultTableModel) jTable.getModel();
    model.addRow(new Object[]{"0", "", "", ""});
}

//Este método retorna a quantidade de novos existentes na tabela Serviços
mostrados na
//interface do formulário.
public int quantRegistrosNovos (JTable jTable)
{
    int qrn = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0")) qrn++;
    }
    return qrn;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
public void preencheRegistrosNovos (Servico[] serv, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0"))
        {
            serv[j].setStatus(jTable.getValueAt(i, 1).toString());
            serv[j].setHost(jTable.getValueAt(i, 2).toString());
            serv[j].setPorta(Integer.parseInt(jTable.getValueAt(i,
3).toString()));
            j++;
        }
    }
}

//Este método retorna a quantidade de registros já existentes na tabela
Atividade mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int qra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0")) qra++;
    }
    return qra;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Servico[] serv, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (!jTable.getValueAt(i, 0).equals("0"))
        {
            serv[j].setCodigo(Integer.parseInt(jTable.getValueAt(i,
0).toString()));

```

```

serv[j].setStatus(jTable.getValueAt(i, 1).toString());
serv[j].setHost(jTable.getValueAt(i, 2).toString());
serv[j].setPorta(Integer.parseInt(jTable.getValueAt(i,
3).toString()));
        j++;
    }
}

//Este método apenas retorna verdadeiro se existe uma linha selecionada na
tabela da interface
//do formulário principal.
public boolean existeLinhaSelecionada (JTable jTable)
{
    if (jTable.getSelectedRow() == -1) return false;
    return true;
}

//Este método preenche um objeto servico com os dados da linha selecionada na
tabela da interface
//do formulário principal.
public void getRegistroSelecionado (Servico serv, JTable jTable)
{
    int l = jTable.getSelectedRow();
    serv.setCodigo (Integer.parseInt(jTable.getValueAt(l, 0).toString()));
    serv.setStatus(jTable.getValueAt(l, 1).toString());
    serv.setHost(jTable.getValueAt(l, 2).toString());
    serv.setPorta(Integer.parseInt(jTable.getValueAt(l, 3).toString()));
}
}

```

Classe *InterfaceManterSistema.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/*
/* Esta classe é responsável por manter a interface da tabela Sistema */
/* *****/

package interface_usuario;

import java.awt.Dimension;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.*;
import modelo.Sistema;

public class InterfaceManterSistema {

//Este método preenche um vetor de objetos sistema com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
public void preencheTabelaSistema(Sistema[] s, JTable jTable)
{
    TableModel modelo = new DefaultTableModel(s.length,5){
        public boolean isCellEditable(int rowIndex, int mColIndex) {

```

```

        if (mColIndex == 0) return false; else return true;
    }
};

 jTable.setModel(modelo);
 jTable.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
 jTable.setPreferredSize(new Dimension(510, 170+16*s.length));
 jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
 jTable.getColumnModel().getColumn(0).setHeaderValue("Codigo");
 jTable.getColumnModel().getColumn(1).setHeaderValue("Nome");
 jTable.getColumnModel().getColumn(2).setHeaderValue("Status");
 jTable.getColumnModel().getColumn(3).setHeaderValue("Data Inicial");
 jTable.getColumnModel().getColumn(4).setHeaderValue("Data Final");
 for(int i = 0;i<s.length;i++)
 {
     jTable.setValueAt(s[i].getCodigo(), i, 0);
     jTable.setValueAt(s[i].getNome(), i, 1);
     jTable.setValueAt(s[i].getStatus(), i, 2);
     jTable.setValueAt(s[i].getDataInicio(), i, 3);
     jTable.setValueAt(s[i].getDataFim(), i, 4);
 }
}

//Método responsável por criar uma linha nova em branco na tabela da interface do
//formulário
//principal.
public void criaLinhaTabelaSistema(JTable jTable)
{
    DefaultTableModel model = (DefaultTableModel) jTable.getModel();
    model.addRow(new Object[]{"0", "", "", "", ""});
}

//Este método retorna a quantidade de registros novos na tabela Sistema mostrados
na
//interface do formulário.
public int quantRegistrosNovos (JTable jTable)
{
    int qrn = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0")) qrn++;
    }
    return qrn;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros.
public void preencheRegistrosNovos (Sistema[] s, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
    {
        if (jTable.getValueAt(i, 0).equals("0"))
        {
            s[j].setNome(jTable.getValueAt(i, 1).toString());
            s[j].setStatus(jTable.getValueAt(i, 2).toString());
            s[j].setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(i,
3).toString()));
            s[j].setDataFim(java.sql.Date.valueOf(jTable.getValueAt(i,
4).toString()));
            j++;
        }
    }
}
}

```

```

//Este método retorna a quantidade de registros já existentes na tabela Atividade
mostrados na
//interface do formulário.
public int quantRegistrosAntigos (JTable jTable)
{
    int qra = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
        {
            if (!jTable.getValueAt(i, 0).equals("0")) qra++;
        }
    return qra;
}

//Este método preenche um vetor de objetos equipe com todas as informações
alteradas ou não
//da tabela da interface do formulário dos registros antigos.
public void preencheRegistrosAntigos (Sistema[] s, JTable jTable)
{
    int j = 0;

    for (int i = 0; i < jTable.getRowCount(); i++)
        {
            if (!jTable.getValueAt(i, 0).equals("0"))
                {
                    s[j].setCodigo(Integer.parseInt(jTable.getValueAt(i,
0).toString()));
                    s[j].setNome(jTable.getValueAt(i, 1).toString());
                    s[j].setStatus(jTable.getValueAt(i, 2).toString());
                    s[j].setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(i,
3).toString()));
                    s[j].setDataFim(java.sql.Date.valueOf(jTable.getValueAt(i,
4).toString()));
                    j++;
                }
        }
}

//Este método apenas retorna verdadeiro se existe uma linha selecionada na tabela
da interface
//do formulário principal.
public boolean existeLinhaSelecionada (JTable jTable)
{
    if (jTable.getSelectedRow() == -1) return false;
    return true;
}

//Este método preenche um objeto sistema com os dados da linha selecionada na
tabela da interface
//do formulário principal.
public void getRegistroSelecionado (Sistema s, JTable jTable)
{
    int l = jTable.getSelectedRow();
    s.setCodigo (Integer.parseInt(jTable.getValueAt(l, 0).toString()));
    s.setNome(jTable.getValueAt(l, 1).toString());
    s.setStatus(jTable.getValueAt(l, 2).toString());
    s.setDataInicio(java.sql.Date.valueOf(jTable.getValueAt(l, 3).toString()));
    s.setDataFim(java.sql.Date.valueOf(jTable.getValueAt(l, 4).toString()));
}
}

```

A.5 PACOTE MODELO

Classe *Atividade.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/*
/* Esta classe é responsável pela definição do objeto da entidade Atividade */
/* *****/

package modelo;

public class Atividade {

    int Codigo;
    String Host;
    String Status;
    String NomeArqArt;

    public Atividade()
    {
    }
    public int getCodigo() {
        return Codigo;
    }
    public void setCodigo(int codigo) {
        Codigo = codigo;
    }
    public String getHost() {
        return Host;
    }
    public void setHost(String host) {
        Host = host;
    }
    public String getStatus() {
        return Status;
    }
    public void setStatus(String status) {
        Status = status;
    }
    public String getNomeArqArt() {
        return NomeArqArt;
    }
    public void setNomeArqArt(String nomeArqArt) {
        NomeArqArt = nomeArqArt;
    }
}
}

```

Classe *CalculaCoeficientes.java*

```

/* *****/
/* Gestor de Software na Nuvem */

```

```

/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pelo cálculo dos coeficientes angular e linear das */
/* equações dos segmentos de reta das funções de pertinência dos termos das */
/* variáveis linguísticas dos sistemas de inferências nebulosos. */
/* *****/

```

```
package modelo;
```

```

public class CalculaCoeficientes {
    public static double Coef_angular;
    public static double Coef_linear;

    public static double Angular ( double mi_esquerdo, double mi_direito, double
intervalo_esquerdo, double intervalo_direito) {
        Coef_angular = (mi_esquerdo - mi_direito) / ( intervalo_esquerdo -
intervalo_direito);
        return Coef_angular;
    }

    public static double Linear ( double mi_esquerdo, double mi_direito, double
intervalo_esquerdo, double intervalo_direito) {
        Coef_angular = Angular ( mi_esquerdo, mi_direito, intervalo_esquerdo,
intervalo_direito);
        Coef_linear = mi_esquerdo - (intervalo_esquerdo * Coef_angular);
        return Coef_linear;
    }
}

```

Classe *DeFuzzy.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe faz a inferência nebulosa, utilizando o método de defuzzyficação */
/* da médiaaritmética ponderada dos centróides pela área. */
/* *****/

```

```
package modelo;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Iterator;

public class DeFuzzy{
    public Connection con = null;
    private String connectionUrl="jdbc:mysql://localhost/lnebulosa";
    private Statement stmt = null;
    private int Cod_regra;

```



```

        int Cod_sistema;
        int Cod_variavel1;
        int Valor_var1;
        int Cod_termo1;
        double Alfa1;
        int Cod_variavel2;
        int Valor_var2;
        int Cod_termo2;
        double Alfa2;
        int Cod_variavel3;
        int Valor_var3;
        int Cod_termo3;
        double Alfa3;
        double Menor_alfa;
        int Cod_Termo_Cons;
        double denominadorRegra = 0;
        double numeradorRegra = 0;
        double ValorFinal = 0;
        ArrayList regras = new ArrayList();
        ArrayList novoAlfa = new ArrayList();

        public DeFuzzy(int cod_sistema, int cod_variavel1, int valor_var1, int
cod_variavel2, int valor_var2, int cod_variavel3, int valor_var3) {
            Cod_sistema = cod_sistema;
            Cod_variavel1 = cod_variavel1;
            Valor_var1 = valor_var1;
            Cod_variavel2 = cod_variavel2;
            Valor_var2 = valor_var2;
            Cod_variavel3 = cod_variavel3;
            Valor_var3 = valor_var3;
        }

        public double calculaAlfa (double coef_linear, double coef_angular, int
valor_var) {
            double alfa = (coef_angular * valor_var) + coef_linear;
            return alfa;
        }

        public void ConnectionBean(){

            try{
                Class.forName("com.mysql.jdbc.Driver");
                con = DriverManager.getConnection(connectionUrl,"root","1234");
                stmt = con.createStatement();
            }
            catch(Exception e){
                System.out.println(e.getMessage());
                con = null;
            }
        }

        public int verificarTermoRegra(int cod_regra, int cod_variavel){
            int cod_termo = 0;
            String sql = "SELECT termo.cod_termo FROM termo, regra, antecedente " +
                "WHERE ( termo.cod_termo = antecedente.cod_termo OR
termo.cod_termo = regra.cod_termo_cons) AND "+
                " regra.cod_regra      = antecedente.cod_regra
AND "+
                " regra.cod_sistema  = "+ Cod_sistema +" AND
"+
                " regra.cod_regra    = "+ cod_regra +" AND
"+
                " termo.cod_variavel = "+ cod_variavel +" ";

            try{
                ConnectionBean ();
                if(con == null){

```

```

        System.out.println("Servidor ocupado.Tente mais tarde!");
    }
    ResultSet rs = stmt.executeQuery(sql);
    if (rs.next()){
        cod_termo = rs.getInt("cod_termo");
    }
    con.close();
    stmt.close();
    return cod_termo;
}
catch (SQLException e) {
    System.out.println(e);
    return -1;
}
catch (Exception e) {
    System.out.println(e);
    return -1;
}
}

public double avaliarMi (int cod_termo, int valor_var){
    int CodSegmentos_termo;
    double alfa = 0;
    String sql = "SELECT cod_segmento, coef_angular, coef_linear FROM segmento
WHERE cod_termo = "+ cod_termo +" AND (intervalo_esquerdo <= "+ valor_var +" ) AND
(intervalo_direito >= "+ valor_var + " ) AND (mi_esquerdo <> 0 OR mi_direito <> 0 )
" ;

    try{
        ConnectionBean ();
        if(con == null){
            System.out.println("Servidor ocupado.Tente mais tarde!");
        }
        ResultSet rs = stmt.executeQuery(sql);
        if(rs.next()){
            CodSegmentos_termo = rs.getInt("cod_segmento");
            double coef_linear = rs.getDouble("coef_linear");
            double coef_angular = rs.getDouble("coef_angular");
            alfa = calculaAlfa(coef_linear, coef_angular,
valor_var);

            return alfa;
        }
        con.close();
        stmt.close();
        return alfa;
    }
    catch (SQLException e) {
        System.out.println(e);
        return -1;
    }
    catch (Exception e) {
        System.out.println(e);
        return -1;
    }
}

public int verificarTermoConsRegra(int cod_sistema, int cod_regra){
    int cod_termo = 0;
    String sql = "SELECT cod_termo_cons FROM regra " +
        "WHERE regra.cod_sistema = "+ Cod_sistema +" AND "+
        " regra.cod_regra = "+ cod_regra +" ";

    try{
        ConnectionBean ();
        if(con == null){
            System.out.println("Servidor ocupado.Tente mais tarde!");

```

```

    }
    ResultSet rs = stmt.executeQuery(sql);
    if (rs.next()){
        cod_termo = rs.getInt("cod_termo_cons");
    }
    con.close();
    stmt.close();
    return cod_termo;
}
catch (SQLException e) {
    System.out.println(e);
    return -1;
}
catch (Exception e) {
    System.out.println(e);
    return -1;
}
}

public double calculaCentroide(double intEsq, double intDir, double coefAng,
double coefLinear) {
    double centroide;
    double potencia1 = Math.pow(intDir, 3);
    double potencia2 = Math.pow(intDir, 2);
    double potencia3 = Math.pow(intEsq, 3);
    double potencia4 = Math.pow(intEsq, 2);
    double aux1 = (potencia1 * coefAng) / 3;
    double aux2 = (potencia2 * coefLinear) / 2;
    double aux3 = (potencia3 * coefAng) / 3;
    double aux4 = (potencia4 * coefLinear) / 2;

    centroide = (aux1 + aux2 - aux3 - aux4);
    return centroide;
}

public double calculaArea(double intEsq, double intDir, double coefAng, double
coefLinear) {
    double area;
    double potencia1 = Math.pow(intDir, 2);
    double potencia3 = Math.pow(intEsq, 2);
    double aux1 = (potencia1 * coefAng) / 2;
    double aux2 = (intDir * coefLinear);
    double aux3 = (potencia3 * coefAng) / 2;
    double aux4 = (intEsq * coefLinear);

    area = (aux1 + aux2 - aux3 - aux4);
    return area;
}

public double gravaTabelaTemp(int codTermoCons) {
    double coefAng;
    double coefLinear;
    double numeradorSegmento;
    double denominadorSegmento;
    double CentroideRegra = 0;
    int retInsere = 0;
    numeradorRegra = 0;
    denominadorRegra = 0;

    String sql = "SELECT cod_segmento, intervalo_esquerdo, intervalo_direito,
mi_esquerdo, mi_direito FROM segmento WHERE cod_termo = "+ codTermoCons + " AND
(mi_esquerdo <> 0 OR mi_direito <> 0 ) ";

    try{

        ConnectionBean ();
        if(con == null){
            System.out.println("Servidor ocupado.Tente mais tarde!");

```

```

        return -1;
    }
    ResultSet rs = stmt.executeQuery(sql);
    while(rs.next()){
        int        codSegmento = rs.getInt("cod_segmento");
        double  intEsq = rs.getDouble("intervalo_esquerdo");
        double  intDir = rs.getDouble("intervalo_direito");
        double  miEsq = rs.getDouble("mi_esquerdo");
        double  miDir = rs.getDouble("mi_direito");
        numeradorSegmento = 0;
        denominadorSegmento = 0;

        if (miEsq == 1) {
            miEsq = Menor_alfa;
        }
        if (miDir == 1) {
            miDir = Menor_alfa;
        }
        coefAng = CalculaCoeficientes.Angular ( miEsq, miDir,
intEsq, intDir);
        coefLinear = CalculaCoeficientes.Linear ( miEsq, miDir,
intEsq, intDir);
        //retInsere = insereTabelaTemp(codTermoCons, codSegmento,
intEsq, intDir, coefAng, coefLinear, miEsq, miDir);
        //if (retInsere <= 0){
        //    return retInsere;
        //}

        numeradorSegmento = calculaCentroide (intEsq, intDir, coefAng,
coefLinear);
        denominadorSegmento = calculaArea (intEsq, intDir, coefAng,
coefLinear);

        numeradorRegra          = numeradorRegra +
(numeradorSegmento);
        denominadorRegra      = denominadorRegra + (denominadorSegmento);

    }
    CentroideRegra = numeradorRegra / denominadorRegra;
    con.close();
    stmt.close();
    return CentroideRegra;
}
catch (SQLException e) {
    System.out.println(e.getMessage());
    return -1;
}
catch (Exception e) {
    System.out.println(e.getMessage());
    return -1;
}
}

public double percorrerRegras(){
    double NumeradorGeral = 0;
    double DenominadorGeral = 0;
    double centroideRegra = 0;

    String sql = "SELECT cod_sistema, cod_regra, cod_termo_cons FROM regra
WHERE cod_sistema = " + Cod_sistema + " ";

    try{
        ConnectionBean ();
        if(con == null){
            System.out.println("Servidor ocupado.Tente mais tarde!");
            return -1;
        }
    }
}

```

```

ResultSet rs = stmt.executeQuery(sql);

while(rs.next()){
    centroideRegra = 0;
    denominadorRegra = 0;

    Cod_regra = rs.getInt("cod_regra");
//System.out.println(Cod_regra);

    Cod_termo1 = verificarTermoRegra ( Cod_regra, Cod_variavel1
);
        if (Cod_termo1 > 0) {
            Alfa1 = avaliarMi(Cod_termo1, Valor_var1);

            if (Alfa1 < 0) {
                System.out.println("ERRO DO ALFA 1");
                return -1;
            }
        }
        else{
            System.out.println("ERRO DO TERMO 1");
            return -1;
        }

    Cod_termo2 = verificarTermoRegra ( Cod_regra, Cod_variavel2
);
    if (Cod_termo2 > 0) {
        Alfa2 = avaliarMi(Cod_termo2, Valor_var2);

        if (Alfa2 < 0) {
            System.out.println("ERRO DO ALFA 2");
            return -1;
        }
    }
    else{
        System.out.println("ERRO DO TERMO 2");
        return -1;
    }

    Cod_termo3 = verificarTermoRegra ( Cod_regra, Cod_variavel3
);
    if (Cod_termo3 > 0) {
        Alfa3 = avaliarMi(Cod_termo3, Valor_var3);

        if (Alfa3 < 0) {
            System.out.println("ERRO DO ALFA 3");
            return -1;
        }
    }
    else{
        System.out.println("ERRO DO TERMO 3");
        return -1;
    }
    if (Alfa1 <= Alfa2){
        Menor_alfa = Alfa1;
    }
    else{
        Menor_alfa = Alfa2;
    }
    if (Alfa3 <= Menor_alfa)
        Menor_alfa = Alfa3;
    if (Menor_alfa > 0){

        Cod_Termo_Cons = verificarTermoConsRegra(Cod_sistema,
Cod_regra );
        centroideRegra = gravaTabelaTemp(Cod_Termo_Cons);
        regras.add(new Integer(Cod_regra));
    }
}
}

```

```

        novoAlfa.add(new Double(Menor_alfa));

        //if (retgrava <= 0 ){
        //    System.out.println("ERRO AO INSERIR NA TABELA
TEMPORÁRIA");
        //    return -1;
        //}

        NumeradorGeral = NumeradorGeral + (centroideRegra *
denominadorRegra);
        DenominadorGeral = DenominadorGeral + denominadorRegra;
    }
    }
    ValorFinal = NumeradorGeral / DenominadorGeral;
    con.close();
    stmt.close();
    return ValorFinal;
}
catch (SQLException e) {
    System.out.println(e);
    return -1;
}
catch (Exception e) {
    System.out.println(e);
    return -1;
}
}
public Iterator getRegras() {
    Iterator Regras = regras.iterator();
    return Regras;
}
public Iterator getNovoAlfa() {
    Iterator Alfas = novoAlfa.iterator();
    return Alfas;
}
}

```

Classe *DeFuzzyBean.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade de inferência */
/* nebulosa defuzzy. */
/* *****/

package modelo;

import java.util.ArrayList;
import java.util.Iterator;

public class DeFuzzyBean{
    public int Cod_sistema;
        int Cod_variavel1;
        int Cod_variavel2;
        int Cod_variavel3;
        int Valor_var1;
        int Valor_var2;
        int Valor_var3;
        double resultadoFinal;
}

```

```

        int Cod_Termo_Cons;
        DeFuzzy des;
        private ArrayList regras;
        private ArrayList novoAlfa;

    public double getResultadoFinal() {
        return resultadoFinal;
    }
    public void setResultadoFinal() {
        des = new DeFuzzy(Cod_sistema, Cod_variavel1, Valor_var1, Cod_variavel2,
Valor_var2, Cod_variavel3, Valor_var3);
        double resFinal = des.percorrerRegras();
        resultadoFinal = resFinal;
    }
    public int getCod_sistema() {
        return Cod_sistema;
    }
    public void setCod_sistema(int cod_sistema) {
        Cod_sistema = cod_sistema;
    }
    public int getCod_variavel1() {
        return Cod_variavel1;
    }
    public void setCod_variavel1(int cod_variavel1) {
        Cod_variavel1 = cod_variavel1;
    }
    public int getCod_variavel2() {
        return Cod_variavel2;
    }
    public void setCod_variavel2(int cod_variavel2) {
        Cod_variavel2 = cod_variavel2;
    }
    public int getCod_variavel3() {
        return Cod_variavel3;
    }
    public void setCod_variavel3(int cod_variavel3) {
        Cod_variavel3 = cod_variavel3;
    }
    public int getValor_var1() {
        return Valor_var1;
    }
    public void setValor_var1(int valor_var1) {
        Valor_var1 = valor_var1;
    }
    public int getValor_var2() {
        return Valor_var2;
    }
    public void setValor_var2(int valor_var2) {
        Valor_var2 = valor_var2;
    }
    public int getValor_var3() {
        return Valor_var3;
    }
    public void setValor_var3(int valor_var3) {
        Valor_var3 = valor_var3;
    }
    public int getCod_Termo_Cons() {
        int teste = des.Cod_Termo_Cons;
        return teste;
    }
    /*
    public Iterator getRegras() {
        Iterator Regras = des.getRegras();
        return Regras;
    }
    public Iterator getNovoAlfa() {
        Iterator Alfas = des.getNovoAlfa();
        return Alfas;
    }

```

```

    }
    */
}

```

Classe *Equipe.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Equipe. */
/* *****/

package modelo;

public class Equipe {

    int codigo_funcionario;
    String nome_funcionario;
    String cargo_funcionario;
    int codigo_fase;

    public Equipe () {

    }

    public int getCodigo_funcionario() {
        return codigo_funcionario;
    }

    public void setCodigo_funcionario(int codigoFuncionario) {
        codigo_funcionario = codigoFuncionario;
    }

    public String getNome_funcionario() {
        return nome_funcionario;
    }

    public void setNome_funcionario(String nomeFuncionario) {
        nome_funcionario = nomeFuncionario;
    }

    public String getCargo_funcionario() {
        return cargo_funcionario;
    }

    public void setCargo_funcionario(String cargoFuncionario) {
        cargo_funcionario = cargoFuncionario;
    }

    public int getCodigo_fase() {
        return codigo_fase;
    }

    public void setCodigo_fase(int codigoFase) {
        codigo_fase = codigoFase;
    }

}

```


Classe *Fase.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Fase */
/*****/

package modelo;

public class Fase {

    int Codigo;
    String Nome;
    String Host;
    String Status;
    java.sql.Date DataInicio;
    java.sql.Date DataFim;
    int CodigoSistema;

    public Fase ()
    {
    }

    public int getCodigo() {
        return Codigo;
    }
    public void setCodigo(int codigo) {
        Codigo = codigo;
    }
    public String getNome() {
        return Nome;
    }
    public void setNome(String nome) {
        Nome = nome;
    }
    public String getHost() {
        return Host;
    }
    public void setHost(String host) {
        Host = host;
    }
    public String getStatus() {
        return Status;
    }
    public void setStatus(String status) {
        Status = status;
    }
    public java.sql.Date getDataInicio() {
        return DataInicio;
    }
    public void setDataInicio(java.sql.Date dataInicio) {
        DataInicio = dataInicio;
    }
    public java.sql.Date getDataFim() {
        return DataFim;
    }
    public void setDataFim(java.sql.Date dataFim) {

```

```

        DataFim = dataFim;
    }
    public int getCodigoSistema() {
        return CodigoSistema;
    }
    public void setCodigoSistema(int codigoSistema) {
        CodigoSistema = codigoSistema;
    }
}

```

Classe *Host.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Host */
/* *****/

package modelo;

public class Host {

    String IP;
    int Servicos;
    double Atraso;

    public String getIP() {
        return IP;
    }
    public void setIP(String iP) {
        IP = iP;
    }
    public int getServicos() {
        return Servicos;
    }
    public void setServicos(int servicos) {
        Servicos = servicos;
    }
    public double getAtraso() {
        return Atraso;
    }
    public void setAtraso(double atraso) {
        Atraso = atraso;
    }
}

```

Classe *Monitoramento.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */

```

```

/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Monitoramento */
/*****

package modelo;

public class Monitoramento {

    int Codigo;
    String Host;
    double Atraso;
    int Servicos;
    int Tentativas;
    double IndiceProbRede;
    int ArtefatosErrados;
    int ArtefatosAtrasados;
    double IndiceProbSeg;
    public int getCodigo() {
        return Codigo;
    }
    public void setCodigo(int codigo) {
        Codigo = codigo;
    }
    public String getHost() {
        return Host;
    }
    public void setHost(String host) {
        Host = host;
    }
    public double getAtraso() {
        return Atraso;
    }
    public void setAtraso(double atraso) {
        Atraso = atraso;
    }
    public int getServicos() {
        return Servicos;
    }
    public void setServicos(int servicos) {
        Servicos = servicos;
    }
    public int getTentativas() {
        return Tentativas;
    }
    public void setTentativas(int tentativas) {
        Tentativas = tentativas;
    }
    public double getIndiceProbRede() {
        return IndiceProbRede;
    }
    public void setIndiceProbRede(double indiceProbRede) {
        IndiceProbRede = indiceProbRede;
    }
    public int getArtefatosErrados() {
        return ArtefatosErrados;
    }
    public void setArtefatosErrados(int artefatosErrados) {
        ArtefatosErrados = artefatosErrados;
    }
    public int getArtefatosAtrasados() {
        return ArtefatosAtrasados;
    }
    public void setArtefatosAtrasados(int artefatosAtrasados) {
        ArtefatosAtrasados = artefatosAtrasados;
    }
    public double getIndiceProbSeg() {
        return IndiceProbSeg;
    }
}

```

```

}
public void setIndiceProbSeg(double indiceProbSeg) {
    IndiceProbSeg = indiceProbSeg;
}

}

```

Classe *Servico.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Servico */
/*****/

```

```
package modelo;
```

```

public class Servico {

    int Codigo;
    String Status;
    String Host;
    int Porta;
    public int getCodigo() {
        return Codigo;
    }
    public void setCodigo(int codigo) {
        Codigo = codigo;
    }
    public String getStatus() {
        return Status;
    }
    public void setStatus(String status) {
        Status = status;
    }
    public String getHost() {
        return Host;
    }
    public void setHost(String host) {
        Host = host;
    }
    public int getPorta() {
        return Porta;
    }
    public void setPorta(int porta) {
        Porta = porta;
    }
}

```

Classe *Sistema.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */

```

```

/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Sistema */
/*****/

package modelo;

public class Sistema {

    int Codigo;
    String Nome;
    String Status;
    java.sql.Date DataInicio;
    java.sql.Date DataFim;

    public Sistema() {
    }

    public String getNome() {
        return Nome;
    }

    public void setNome(String nome) {
        Nome = nome;
    }

    public String getStatus() {
        return Status;
    }

    public void setStatus(String status) {
        Status = status;
    }

    public java.sql.Date getDataInicio() {
        return DataInicio;
    }

    public void setDataInicio(java.sql.Date dataInicio) {
        DataInicio = dataInicio;
    }

    public java.sql.Date getDataFim() {
        return DataFim;
    }

    public void setDataFim(java.sql.Date dataFim) {
        DataFim = dataFim;
    }

    public int getCodigo() {
        return Codigo;
    }

    public void setCodigo(int codigo) {
        Codigo = codigo;
    }

}

```

APÊNDICE B — FONTES DO NODO DE SOFTWARE NA NUVEM

B.1 PACOTE CONTROLE

Classe *ControlaNodo.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pelo acionamento das funcionalidades do nodo, */
/* promovendo a interação entre os pacotes de interface e o pacote modelo. */
/*****/

package controle;

import interface_rede.Servidor;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import modelo.*;

public class ControlaNodo {

    Gestor gest = new Gestor();
    int MinhaPorta = -1;
    Servidor serv;

    //Método responsável por obter um novo valor de porta do usuário e defini-lo para o
    nodo
    //Em seguida, uma nova instância de servidor é criada com este novo valor de porta
    public void alterarPorta (JLabel jLabelPorta, JLabel jLabelrecepcao)
    {
        String p = JOptionPane.showInputDialog(jLabelPorta, "Defina a Minha Porta:",
            "Entrada de dado", JOptionPane.QUESTION_MESSAGE);
        if (!p.equals(null))
        {
            MinhaPorta = (Integer.parseInt(p));
            jLabelPorta.setText("VALOR DA MINHA PORTA: " + MinhaPorta);
            if (serv != null) serv.setListening(false);
            serv = new Servidor(MinhaPorta, jLabelPorta, jLabelrecepcao);
            serv.start();
        }
    }

    //Método responsável por obter um novo valor de porta do usuário e defini-lo para o
    Gestor
    public void alterarPortaGestor (JLabel jLabelIDgestor)
    {
        String p = JOptionPane.showInputDialog(jLabelIDgestor, "Porta do Gestor:",
            "Entrada de dado", JOptionPane.QUESTION_MESSAGE);
        gest.setPorta(Integer.parseInt(p));
        jLabelIDgestor.setText("IDENTIFICAÇÃO DO GESTOR: Porta = " +
            p + " e Endereço IP = " + gest.getIP());
    }

    //Método responsável por obter um novo valor de IP do usuário e defini-lo para o
    Gestor
    public void alterarIPGestor (JLabel jLabelIDgestor)

```

```

{
    String ip = JOptionPane.showInputDialog(jlabelIDgestor, "Endereço IP do
Gestor:",
        "Entrada de dado", JOptionPane.QUESTION_MESSAGE);
    gest.setIP(ip);
    jlabelIDgestor.setText("IDENTIFICAÇÃO DO GESTOR: Porta = " +
        gest.getPorta() + " e Endereço IP = " +
gest.getIP());
}

public int obterPortaGestorAtual()
{
    return gest.getPorta();
}

public String obterIPGestorAtual()
{
    return gest.getIP();
}
}

```

B.2 PACOTE INTERFACE_REDE

Classe *Cliente.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela iniciativa da comunicação por parte do nodo */
/* para enviar um artefato para o Gestor. */
/* *****/

package interface_rede;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import javax.swing.*;

public class Cliente extends Thread {

    JLabel jlabel;
    JProgressBar jprogressbar;
    String nomeArqArt;
    int PortaGestor;
    String IPGestor;

    public Cliente (JLabel jLabelEnviaArt,
        JProgressBar jProgBarEnviaArt, String outFile,
        int porta, String IP)
    {

```

```

        jlabel = jLabelEnviaArt;
        jprogressbar = jProgBarEnviaArt;
        nomeArqArt = outFile;
        PortaGestor = porta;
        IPGestor = IP;
    }

    //Método responsável por enviar o arquivo do artefato propriamente dito.
    public void transfereArtefato (Socket socketcliente, JLabel jLabelEnviaArt,
                                   JProgressBar jProgBarEnviaArt,
                                   String outFile)
    {
        jLabelEnviaArt.setText(jLabelEnviaArt.getText() + " " + outFile);
        File file = new File(outFile);
        try {
            DataInputStream input = new DataInputStream(new FileInputStream(file));
            DataOutputStream outputMsg = new
            DataOutputStream(socketcliente.getOutputStream());

            byte[] cache = new byte[10240];
            int size = 0;
            long fileSize = file.length();
            jProgBarEnviaArt.setMinimum(0);
            jProgBarEnviaArt.setMaximum((int)fileSize);
            jProgBarEnviaArt.setValue(0);
            while ((size = input.read(cache)) > -1) {
                outputMsg.write(cache, 0, size);
                jProgBarEnviaArt.setValue(jProgBarEnviaArt.getValue() + size);
            }
            outputMsg.close();
            input.close();
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    //Método responsável por fazer a negociação da transmissão com o Gestor para dar
    início ao
    //envio do arquivo do artefato.
    public void enviaArtefato(JLabel jLabelEnviaArt,
                              JProgressBar jProgBarEnviaArt, String outFile)
    {
        InetAddress ia = null;
        try {
            ia = InetAddress.getLocalHost();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }

        try {
            Socket socketcliente = new Socket(IPGestor,PortaGestor);
            DataOutputStream outSDU = new
            DataOutputStream(socketcliente.getOutputStream());

            outSDU.writeUTF(ia.getHostAddress());
            Protocolo prot = new Protocolo();
            prot.setComandoUsuario("Enviar Artefato");
            prot.executaIniciador();
            if (prot.getPrimitiva().equals("ENVIAARTrequest"))
            {
                outSDU.writeUTF(prot.getPDU());

            transfereArtefato(socketcliente,jLabelEnviaArt,jProgBarEnviaArt,outFile);
                prot.setPrimitiva("FIMARTconfirmation");
                prot.executaIniciador();
            }
            outSDU.close();
            socketcliente.close();
        }
    }

```



```

        JOptionPane.showMessageDialog(null, "Artefato enviado com sucesso.",
            "Nodo de Software na Nuvem - Artefato",
JOptionPane.INFORMATION_MESSAGE);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.toString(),
            "Nodo de Software na Nuvem - Artefato",
JOptionPane.INFORMATION_MESSAGE);
    }

    jLabelEnviaArt.setVisible(false);
    jProgBarEnviaArt.setVisible(false);
}

public void run()
{
    enviaArtefato (jlabel, jprogressbar, nomeArqArt);
}
}

```

Classe *Protocolo.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável por executar o protocolo de rede. */
/* *****/
/*****/

package interface_rede;

public class Protocolo {

    public static enum tipoEstado {REPOUSO, ENVIANDOART, RESPONDENDOPING};
    private tipoEstado estado = tipoEstado.REPOUSO;
    private String PDU = null;
    private String Primitiva = null;
    private String ComandoUsuario = null;

    public tipoEstado getEstado() {
        return estado;
    }

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

    public String getPDU() {
        return PDU;
    }

    public void setPDU(String pdu) {
        PDU = pdu;
    }

    public String getPrimitiva() {
        return Primitiva;
    }

    public void setPrimitiva(String primitiva) {
        Primitiva = primitiva;
    }
}

```

```

}

public String getComandoUsuario() {
    return ComandoUsuario;
}

public void setComandoUsuario(String comandoUsuario) {
    ComandoUsuario = comandoUsuario;
}

//Este método é responsável pelo processamento do lado respondedor do protocolo.
public void executaRespondedor() {

    switch (estado)
    {
        case REPOUSO:
            if (PDU.equals("PING") && Primitiva.equals("PINGindicacion"))
                { estado = tipoEstado.RESPONDENDOPING; Primitiva =
"PONGrequest"; PDU = "PONG";}
                break;
            case RESPONDENDOPING:
                if (Primitiva.equals("PONGconfirmation"))
                    {estado = tipoEstado.REPOUSO; Primitiva = null; PDU = null;}
    }
}

//Este método é responsável pelo processamento do lado iniciador do protocolo.
public void executaIniciador() {

    switch (estado)
    {
        case REPOUSO:
            if (ComandoUsuario.equals("Enviar Artefato"))
                { estado = tipoEstado.ENVIANDOART; Primitiva =
"ENVIAARTrequest"; PDU = "ENVIAART";}
                break;
            case ENVIANDOART:
                if (Primitiva.equals("FIMARTconfirmation"))
                    {estado = tipoEstado.REPOUSO; Primitiva = null; PDU = null;}
                break;
    }
}
}

```

Classe *Servidor.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável por aguardar a conexão do Gestor. */
/* *****/
/*****/

package interface_rede;

import java.awt.Color;
import java.io.DataInputStream;

```

```

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.swing.JOptionPane;
import javax.swing.JLabel;
import java.util.*;
import java.text.SimpleDateFormat;

public class Servidor extends Thread {

    int MinhaPorta;
    boolean listening = true;
    JLabel jLabelMinhaPorta;
    JLabel jLabelrecepcao;

    public boolean isListening() {
        return listening;
    }

    public void setListening(boolean listening) {
        this.listening = listening;
    }

    public Servidor (int p, JLabel jLabelMinhaPorta, JLabel jLabelrecepcao)
    {
        MinhaPorta = p;
        this.jLabelMinhaPorta = jLabelMinhaPorta;
        this.jLabelrecepcao = jLabelrecepcao;
    }

    public int getMinhaPorta() {
        return MinhaPorta;
    }

    public void setMinhaPorta(int minhaPorta) {
        MinhaPorta = minhaPorta;
    }

    //Método responsável por aguardar a conexão com o Gestor, executando o protocolo de
    rede
    //para processar a solicitação recebida.
    public void run() {

        ServerSocket serverSocket = null;

        while (listening) {
            try {
                serverSocket = new ServerSocket(MinhaPorta);
            } catch (IOException e) {
                jLabelMinhaPorta.setText("VALOR DA MINHA PORTA: nulo");
                jLabelrecepcao.setText("Esperando recepção...");
                JOptionPane.showMessageDialog(null, "Porta já em uso. \n Escolha
outra.",
                    "Abrir Serviço", JOptionPane.INFORMATION_MESSAGE);
            }
            try {
                serverSocket.close();
            } catch (IOException e1) {System.err.println("falhou fechar a
porta");}
        }

        Socket socket = null;
        try {
            socket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept falhou.");
            System.exit(1);
        }
    }
}

```

```

    try {
        DataInputStream inSDU = new DataInputStream(socket.getInputStream());
        DataOutputStream outSDU = new
DataOutputStream(socket.getOutputStream());
        Protocolo prot = new Protocolo();
        prot.setPDU(inSDU.readUTF());
        prot.setPrimitiva("PINGindication");
        prot.executaRespondedor();
        if (prot.getPrimitiva().equals("PONGrequest"))
        {
            outSDU.writeUTF(prot.getPDU());
            prot.setPrimitiva("PONGconfirmation");
            prot.executaRespondedor();
        }
        socket.close();
        serverSocket.close();
//calendário com data e hora ocidental
GregorianCalendar calendar = new GregorianCalendar();
//pega o tempo (data e hora)
Date data = calendar.getTime();
//Formatando
SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
formato.format(data);
String msg = "Nodo calibrado ou monitorado em: ";
msg = msg + data;
if (jlabelrecepcao.getForeground().equals(Color.red))
    { jlabelrecepcao.setForeground(Color.blue); }
else { jlabelrecepcao.setForeground(Color.red); }
jlabelrecepcao.setText(msg);
    } catch (IOException e) {
        e.printStackTrace();
        jlabelrecepcao.setText("Esperando recepção...");
        try {
            socket.close();
            serverSocket.close(); }
        catch (IOException e1) {System.err.println("falhou fechar a porta");}
    }
}
}
}

```

B.3 PACOTE INTERFACE_USUARIO

Classe *InterfacePrincipal.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável por executar a interface principal */
/* *****/
package interface_rede;

```

```

import java.awt.Color;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.swing.JOptionPane;
import javax.swing.JLabel;
import java.util.*;
import java.text.SimpleDateFormat;

public class Servidor extends Thread {

    int MinhaPorta;
    boolean listening = true;
    JLabel jLabelMinhaPorta;
    JLabel jLabelrecepcao;

    public boolean isListening() {
        return listening;
    }

    public void setListening(boolean listening) {
        this.listening = listening;
    }

    public Servidor (int p, JLabel jLabelMinhaPorta, JLabel jLabelrecepcao)
    {
        MinhaPorta = p;
        this.jLabelMinhaPorta = jLabelMinhaPorta;
        this.jLabelrecepcao = jLabelrecepcao;
    }

    public int getMinhaPorta() {
        return MinhaPorta;
    }

    public void setMinhaPorta(int minhaPorta) {
        MinhaPorta = minhaPorta;
    }

    //Método responsável por aguardar a conexão com o Gestor, executando o protocolo de
    rede
    //para processar a solicitação recebida.
    public void run() {

        ServerSocket serverSocket = null;

        while (listening) {
            try {
                serverSocket = new ServerSocket(MinhaPorta);
            } catch (IOException e) {
                jLabelMinhaPorta.setText("VALOR DA MINHA PORTA: nulo");
                jLabelrecepcao.setText("Esperando recepção...");
                JOptionPane.showMessageDialog(null, "Porta já em uso. \n Escolha
                outra.",
                    "Abrir Serviço", JOptionPane.INFORMATION_MESSAGE);
            }
            try {
                serverSocket.close();
            } catch (IOException e1) {System.err.println("falhou fechar a
            porta");}
        }

        Socket socket = null;
        try {
            socket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept falhou.");
        }
    }
}

```

```

        System.exit(1);
    }

    try {
        DataInputStream inSDU = new DataInputStream(socket.getInputStream());
        DataOutputStream outSDU = new
DataOutputStream(socket.getOutputStream());
        Protocolo prot = new Protocolo();
        prot.setPDU(inSDU.readUTF());
        prot.setPrimitiva("PINGindicacao");
        prot.executaRespondedor();
        if (prot.getPrimitiva().equals("PONGrequest"))
        {
            outSDU.writeUTF(prot.getPDU());
            prot.setPrimitiva("PONGconfirmacao");
            prot.executaRespondedor();
        }
        socket.close();
        serverSocket.close();
    //calendário com data e hora ocidental
    GregorianCalendar calendar = new GregorianCalendar();
    //pega o tempo (data e hora)
    Date data = calendar.getTime();
    //Formatando
    SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
    formato.format(data);
    String msg = "Nodo calibrado ou monitorado em: ";
    msg = msg + data;
    if (jlabelrecepcao.getForeground().equals(Color.red))
        { jlabelrecepcao.setForeground(Color.blue); }
    else { jlabelrecepcao.setForeground(Color.red); }
    jlabelrecepcao.setText(msg);
    } catch (IOException e) {
        e.printStackTrace();
        jlabelrecepcao.setText("Esperando recepção...");
        try {
            socket.close();
            serverSocket.close(); }
        catch (IOException e1) {System.err.println("falhou fechar a porta");}
    }
}
}
}

```

B.4 PROTOCOLO MODELO

Classe *Gestor.java*

```

/* *****/
/* Gestor de Software na Nuvem */
/* Autor: Carolina Y. Ji */
/* Universidade do Estado do Rio de Janeiro */
/* Programa de Pós Graduação em Engenharia Eletrônica */
/* Dissertação de Mestrado */
/* Lógica nebulosa aplicada a um sistema de detecção de intrusos em computação */
/* em nuvem */
/* Orientador: Nival Nunes de Almeida */
/* Coorientador: Orlando Bernardo Filho */
/* Esta classe é responsável pela definição do objeto da entidade Gestor. */
/* *****/
/*****

```

```
package modelo;

public class Gestor {

    int Porta = 4445;
    String IP = "localhost";
    public int getPorta() {
        return Porta;
    }
    public void setPorta(int porta) {
        Porta = porta;
    }
    public String getIP() {
        return IP;
    }
    public void setIP(String iP) {
        IP = iP;
    }
}
```