UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UM *FRAMEWORK* BASEADO EM GRUPOS DE AGENTES DE SOFTWARE ESPECIALIZADOS PARA CONSTRUÇÃO DE SISTEMAS DISTRIBUÍDOS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

LUIZ FERNANDO SIROTHEAU SERIQUE JUNIOR

ORIENTADOR: RAFAEL TIMÓTEO DE SOUSA JÚNIOR

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGENE.DM - 260/06
BRASÍLIA/DF: MAIO - 2006

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UM *FRAMEWORK* BASEADO EM GRUPOS DE AGENTES DE SOFTWARE ESPECIALIZADOS PARA CONSTRUÇÃO DE SISTEMAS DISTRIBUÍDOS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

LUIZ FERNANDO SIROTHEAU SERIQUE JUNIOR

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Prof. Rafael Timóteo de Sousa Júnior, Docteur (ENE-UnB)
(Orientador)

Prof. Anderson C. A. Nascimento, Ph.D (ENE-UnB)
(Examinador Interno)

Prof, Jacir Luiz Bordim, Ph.D (CIC-UnB)
(Examinador Externo)

Prof. Ricardo Staciarini Puttini, Doutor (ENE-UnB)
(Suplente)

FICHA CATALOGRÁFICA

SERIQUE JUNIOR, LUIZ FERNANDO SIROTHEAU SERIQUE.
Um *Framework* baseado em Grupos de Agentes de Software Especializados para Construção de Sistemas Distribuídos de Detecção de Intrusão em Redes de Computadores [Distrito Federal] 2006.
xiv, 177p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2006).
Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica.

1. Segurança de redes	2. Detecção de intrusão
3. Agentes de software	4. *Framework* de aplicação
I. ENE/FT/UnB.	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SERIQUE JUNIOR, L. F. S. S. (2006). Um Framework baseado em Grupos de Agentes de Software Especializados para Construção de Sistemas Distribuídos de Detecção de Intrusão em Redes de Computadores. Dissertação de Mestrado, Publicação 260/06, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 177p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Luiz Fernando Sirotheau Serique Junior
TÍTULO DA DISSERTAÇÃO: Um *Framework* baseado em Grupos de Agentes de Software Especializados para Construção de Sistemas Distribuídos de Detecção de Intrusão em Redes de Computadores.

GRAU: Mestre ANO:2006.

Luiz Fernando Sirotheau Serique Junior
SMPW Quadra 04 Conjunto 05 Condomínio 37 Casa 18.
CEP 72010-990 – Brasília – DF - Brasil

AGRADECIMENTOS

Ao meu orientador Dr. Rafael Timóteo de Sousa Júnior, pela oportunidade, pela confiança depositada e pelos conselhos que foram indispensáveis a este trabalho. Também agradeço ao professor Dr. Ricardo Staciarini Puttini que me ajudou na revisão finalização e formatação desse trabalho.

Ao companheiro de graduação e mestrado, Daniel Lyra, que inicialmente me ajudou no tratamento das bases de ataques e treinamento da rede neural artificial.

Agradeço aos velhos amigos - Flávio Garcia, Michael Alves e Vinicius Vasconcelos - que acompanharam todo o curso desse trabalho e também aguardaram ansiosamente por esse dia.

Aos amigos de trabalho do MMA - Marcus Andrey, Fagner Ernesto e Leandro Malaquias - que se interessaram e foram expectadores dessa pesquisa. Especialmente, à Márcia Kamada, pelo seu profissionalismo exemplar, pela compreensão e apoio em todos os momentos, e, principalmente, por sua amizade. E ao Dr. Paulo Henrique Santana que também apostou nesse trabalho.

À minha mãe, Maria de Lourdes, e ao meu pai, Luiz Fernando (*in memorian*), por toda ajuda, dedicação e amor desde meu nascimento que, com certeza, foram essenciais para mais essa conquista. E aos meus irmãos, Marcelo, Ester e Ana.

Em especial, agradeço à minha esposa, Fernanda Leal Sampaio, pelo companheirismo, compreensão e toda a ajuda nos momentos mais difíceis.

Agradeço sobretudo à Deus, por ter me dado a inspiração necessária, paciência e perseverança para que pudesse concluir esse trabalho da melhor forma possível, e, ainda, por ter colocado essas pessoas tão especiais na minha vida.

Luiz Fernando Sirotheau Serique Junior

RESUMO

UM FRAMEWORK BASEADO EM GRUPOS DE AGENTES DE SOFTWARE ESPECIALIZADOS PARA CONSTRUÇÃO DE SISTEMAS DISTRIBUÍDOS DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

Autor: Luiz Fernando Sirotheau Serique Junior

Orientador: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, maio de 2006

O objetivo do presente trabalho é apresentar a especificação, modelagem e prototipagem de um *framework* para construção de sistemas distribuídos de detecção de intrusão com base em grupos de agentes de software especializados que possuem as habilidades de mobilidade e autonomia. Dentre as características almejadas para o *framework* estão a reusabilidade dos componentes básicos, a capacidade de extensão e a manutenibilidade que juntos possibilitarão uma evolução natural do sistema, permitindo a inserção de novos agentes com maiores graus de especialização e novos componentes de serviços, sem a necessidade de paralisação.

A fim de ilustrar e validar as soluções propostas, foram implementados dois cenários de detecção de intrusão aonde os grupos de agentes especializados puderam exercer suas funções e proteger esses ambientes de rede.

ABSTRACT

A FRAMEWORK FOR BUILDING INTRUSION DETECCTION SYSTEMS OVER COMPUTER NETWORKS BASED ON GROUP OF SPECIALIZED SOFTWARE AGENTS

Author: Luiz Fernando Sirotheau Serique Junior

Supervisor: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, may of 2006

The objective of the current work is presenting the specification, modeling and prototyping of a framework for building distributed intrusion detection systems based on groups of specialized agents that are mobile and autonomous. Amongst the required characteristics for the framework, there are the reusability of basic components, the extension capability and the easy maintenance that together make possible a natural evolution of the system, allowing the insertion of new agents with more specialization and new service components, without interruption.

In order to illustrate and validate proposed solutions, the groups of specialized agents had been implemented two scenes of intrusion detention where had been able to do its functions and protecting these network environments.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

ACC	*Agent Communication Channel*
ACL	*Agent Communication Language*
AID	*Agent Identifier*
AMS	*Agent Management Syatem*
API	*Application Program Interface*
AUML	*Agent-based Unified Modeling Language*
CGI	*Commom Gateway Interface*
DF	*Directory Facilitator*
DMZ	*De-Militarized Zone*
DoS	*Deny of Service*
ETL	*Extraction, Transformation and Load*
FIPA	*Foundation for Intelligent Physical Agents*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
IDS	*Intrusion Detection System*
IIOP	*Internet Inter-ORB Protocol*
IP	*Internet Protocol*
JADE	*Java Agent Development Framework*
JOONE	*Java Object Oriented Neural Engine*
JPCAP	*Java Package for Packet Capture*
JVM	*Java Virtual Machine*
LGPL	Lesser General Public License
GPL	General Public License
MAS	Agent Management System
MLP	Multi Layer Perceptrons
MTP	*Message Transport Protocol*
OKBC	Open Knowledge Base Connectivity
OWL	Web Ontology Language
QoS	Quality of Service
RMI	Remote Method Invocation
RNA	Rede Neural Artificial
SDI	Sistema de Detecção de Intrusão
SQL	*Structured Query Language*
SSO	*Single Sign-On*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol / Internet Protocol*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
VPN	*Virtual Private Network*
AID	*Agent Identifier*
W3C	*World Wide Web Consortium*

1 - INTRODUÇÃO

Nos tempos atuais, o bem mais precioso de qualquer organização é a sua informação. Essa é fonte vital para a geração de conhecimento estratégico e para o aumento da competitividade, necessários em nosso acirrado contexto capitalista e globalizado. Nesse sentido, o setor de tecnologia da informação cresce para atender a demanda por produtos e serviços que melhorem os processos de aquisição, armazenamento, recuperação e, principalmente, integração das fontes de informações.

Uma dessas iniciativas foi o desenvolvimento de tecnologias para a interconexão de corporações por meio da Internet que aumentou sensivelmente a velocidade na troca de informações e promoveu uma maior produtividade. Esse tipo de abordagem também se mostrou bastante viável em termos de custos, pois a utilização de uma rede pública se torna mais barata do que a instalação ou locação de linhas dedicadas. Consequentemente, o numero de empresas e de usuários conectados à Internet vem aumentando a cada dia.

Do mesmo modo que há um investimento constante na conectividade, também é crescente a preocupação de todos em relação à segurança nessas redes, visto que as possibilidades de falhas e fraudes são bem maiores com o aumento do número de portas de acesso e de pessoas envolvidas.

Em um cenário onde todo trabalho é efetuado de forma *on-line*, é necessário garantir a autenticidade, integridade e sigilo das informações, pois, uma vez que essas são interceptadas, danificadas ou forjadas, podem causar grandes prejuízos e, até mesmo, comprometer a credibilidade e a vitalidade dessas organizações.

A fim de melhorar a segurança, diversos mecanismos de proteção têm sido implantados nas organizações. Um mecanismo bastante difundido é o *firewall* que representa a primeira barreira entre duas redes. Dessa forma, as conexões que utilizam portas não autorizadas ou desconhecidas são rejeitadas, controlando, assim, o acesso ao ambiente interno.

Outro mecanismo bastante comum é a utilização de técnicas de criptografia e autenticação para garantir um tráfego seguro na comunicação, formando redes seguras conhecidas como VPNs (*Virtual Private Networks)*.

Essas técnicas prevenirem diversos problemas de segurança, porém não bastam para garantir um ambiente seguro, pois, freqüentemente, existem falhas nos próprios sistemas da corporação e, ainda, podem existir ameaças internas dos usuários, como por exemplo, funcionários mal intencionados ou descontentes. Sendo assim, é necessário monitorar todas as ações dos usuários da rede e ainda se proteger contra erros operacionais que também podem trazer prejuízos à organização.

Outro fator importante é que uma estratégia de segurança não garante que o sistema estará protegido perpetuamente. Na maioria dos casos, uma proteção é contornada pelos invasores após uma incansável busca por novas vulnerabilidades nos sistemas. Uma vez detectadas, essas vulnerabilidades deverão ser corrigidas pelos administradores de segurança para que haja o bloqueio de futuras invasões da mesma natureza, porém sempre existirão novas brechas que exigirão novas correções, fechando, assim, um processo cíclico.

Fica claro que existe a necessidade de uma constante monitoração e evolução dos mecanismos de segurança, pois as técnicas de invasão também avançam com o passar do tempo. Essas atividades podem ser excessivamente complexas para os administradores, principalmente, em grandes redes e ambientes heterogêneos, pois apresentam uma infinidade de pontos de monitoria, uma diversidade de serviços, muitas vezes desconhecidos, além de um enorme volume de dados que trafegam pela rede.

Os sistemas de detecção de intrusão foram criados para auxiliarem os administradores no monitoramento da segurança. O objetivo desse sistema é automatizar o processo de captura de dados da rede, a fim de levantar possíveis ameaças por meio de reconhecimento de padrões. O SDI pode estar associado a algum recurso de inteligência artificial que possibilita a classificação entre padrões normais e anômalos. Mais recentemente, estão sendo aplicadas, também, técnicas de *data mining* visando extrair conhecimento subjacente aos dados capturados, que permite a uma maior predição e

adaptação do sistema. Uma vez que é detectada alguma operação anômala ou indevida, o SDI poderá solicitar um procedimento de contra-ataque, como por exemplo, a desativação do serviço alvo de ataque, ativação de regras do *firewall* para bloqueio da conexão suspeita ou simplesmente a notificação para o administrador da rede.

A maioria dos SDIs possui uma arquitetura monolítica, onde um ou mais sensores são instalados em pontos estratégicos da rede a fim de coletarem informações que serão enviadas para um servidor central que fará o processamento e classificação dos dados recebidos. Esse tipo de arquitetura se mostra pouco escalável, não tolerante a falhas, pouco flexível e ainda ineficiente em termos de manutenção evolutiva. Em contrapartida, os SDIs atuais vêm evoluindo no sentido de tornarem suas arquiteturas mais distribuídas para minimizar as fragilidades e limitações conhecidas dos modelos monolíticos. Naturalmente, a tecnologia de agentes de software vem apoiando fortemente a construção dos sistemas distribuídos em geral, pois traz diversos benefícios, principalmente devido às características de mobilidade e autonomia pertinentes dessa arquitetura.

Os diversos tipos de ataques provenientes de usuários internos e externos, a ameaça dos diversos *malwares*, a complexidade para a monitoria e gerenciamento da segurança em diversos pontos da rede, a necessidade de uma evolução natural do sistema de detecção de intrusão e a demanda por características de inteligência, mobilidade e autonomia sugerem uma arquitetura extensível, modular e distribuída. Dessa forma, o uso de agentes de *software* para o desenvolvimento dos SDIs é muito bem-vindo. Adicionalmente, essa abordagem simplifica a implementação por meio de um esquema baseado em delegação de tarefas e ainda permite a configuração e instalação remota de novos componentes no sistema, sem necessidade de paralisação.

Considerando o incentivo de se utilizar uma abordagem de agentes de software, o presente trabalho propõe um *framework* para construção de sistemas de detecção de intrusão utilizando uma arquitetura baseada em grupos de agentes de software especializados. Como produto desse esforço, foi implementado um protótipo que permite a implementação e extensão dessa proposta para atingir as necessidades de segurança de cada ambiente de rede.

Além de servir de base para a construção de um SDI, esse *framework* tem o intuito de oferecer diversas vantagens para os projetistas e desenvolvedores, dentre elas: a flexibilidade para aplicação de diferentes técnicas de detecção e estratégias de operação; a redução no tempo de desenvolvimento por meio do reuso dos serviços fundamentais e das classes básicas dos agentes já implementadas; e a fácil extensão e manutenção devido ao emprego dos modelos orientados a objetos e baseados em agentes de software.

1.1 - MOTIVAÇÃO

Apesar de existirem diversas pesquisas abordando o uso de agentes de *software* na segurança computacional, poucos trabalhos focam no próprio processo de construção dos SDIs, principalmente, sugerindo uma arquitetura extensível, flexível, portável e de fácil manutenção. Foi observado que as pesquisas são mais freqüentes na melhoria das técnicas de detecção e da performance. Além disso, não puderam ser obtidas bibliotecas livres que fornecessem a estrutura necessária para a imediata aplicação de cenários de detecção de intrusão e que exigissem dos desenvolvedores apenas as especificações de alto nível.

Impulsionado por isso, a proposta fornece um ferramental em termos de *framework* e metodologia que possibilite um processo rápido e orientado a agentes para o desenvolvimento de SDIs. Para preparar essa estrutura da melhor forma possível, foi feito um amplo estudo das diversas técnicas de detecção e a tipologia dos SDIs para que esses serviços pudessem ser contemplados, dando, assim, máxima flexibilidade para os projetistas e desenvolvedores.

Alguns trabalhos serviram de auxílio para a concepção do *framework*. Por exemplo, o projeto *Prelude Hybrid IDS* [1] que também sugere uma arquitetura distribuída e a possibilidade de detecção híbrida (rede e *host*), porém sem o uso de agentes. Algumas idéias do CIDF (*Common Intrusion Detection Framework*) [2] serviram como um modelo de referência para uma especificação compatível com outros projetos. E ainda houve a colaboração do modelo genérico de SDI proposto no trabalho de Bace e Mell [3] e do trabalho de Bernardes [4] que apresenta um SDI com arquitetura baseada em camadas de agentes autônomos e móveis.

Em termos de novidades tecnológicas, o *framework* apresenta uma arquitetura diferenciada que consolida um modelo baseado em grupos de agentes especializados, como mostrado na Figura 1.1. Cada grupo de agentes será formado pelos seguintes componentes: agente de coleta, agente de análise, agente de armazenamento, agente de coordenação e agente de reação.

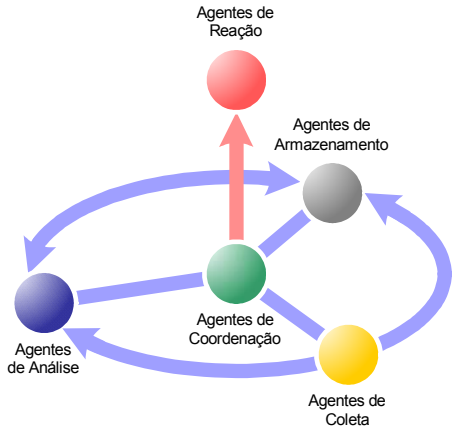


Figura 1.1 - Visão geral do *framework*

Para cada cenário de intrusão, um grupo de agentes poderá ser estendido, conforme mostrado na Figura 1.2, para desempenhar todas as ações, desde a detecção até a reação aos ataques.

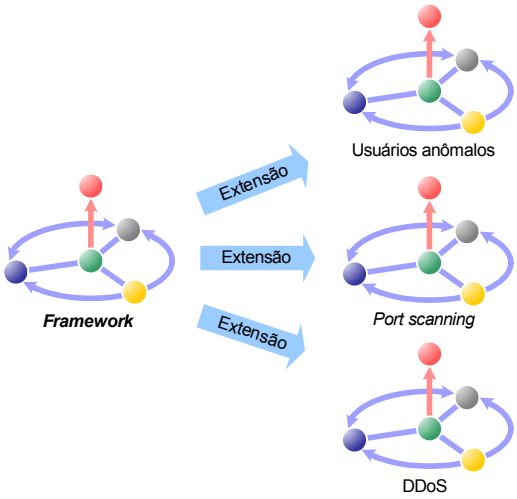


Figura 1.2 - Extensão do framework para diferentes cenários de intrusão

Outros recursos implementados também são de interesse para futuros trabalhos, como a capacidade de importação de ontologias originadas no *software* Protege [5], o componente de rede neural artificial MLP de topologia reajustável em tempo real, o capturador de pacotes TCP/IP, as classes básicas dos grupos de agentes e a interface gráfica móvel para interações com os usuários e administradores.

O que se pretende, também, é difundir o uso do *framework* para os projetos de SDIs de forma que esteja em constante melhoria e possa servir como referência tanto para experimentos, quanto para aplicações reais. Para que isso seja promovido, todo trabalho foi desenvolvido utilizando *software* livre e tecnologias acessíveis.

1.2 - OBJETIVO

O presente trabalho tem por objetivo propor um *framework* para construção de sistemas de detecção de intrusão utilizando uma arquitetura baseada em grupos de agentes de software especializados. Como produto dessa proposta, será implementado um protótipo funcional que já contempla alguns serviços e as classes básicas dos grupos de agentes. Sendo assim, alguns cenários serão validados, como a detecção de usuários anômalos e detecção de *port scanning*.

O *framework* deverá simplificar a implementação e reduzir o tempo de desenvolvimento com a disponibilização de uma arquitetura já preparada com os serviços básicos de interesse e, ainda, permitirá a evolução continua do sistema devido à capacidade de extensão inerente.

1.3 - ESCOPO DO TRABALHO

Para a concepção e implementação do *framework*, foi feito um planejamento prévio das atividades necessárias que foram divididas nas quatro etapas mostradas abaixo.

I) Preparação teórica

- Pesquisa e leitura de trabalhos semelhantes ou de apoio envolvendo segurança computacional, agentes de software e arquiteturas dos sistemas de detecção de intrusão;
- Pesquisa e testes de ferramentas de apoio ao desenvolvimento de sistemas baseados em agentes de software;
- Definição de um ambiente de desenvolvimento baseado em *software* livre.

II) Concepção do *framework*:

- Definição dos grupos de agentes necessários;
- Definição das interações entre os grupos de agentes;
- Definição da arquitetura do *framework;*
- Modelagem dos casos de uso e comportamentos dos agentes básicos usando a extensão AUML (*Agent Unified Modeling Language*);
- Modelagem das classes dos grupos de agentes usando UML (*Unified Modeling Language*).

III) Desenvolvimento do protótipo:

- Implementação das classes básicas dos agentes;
- Criação da ontologia básica do *framework;*
- Implementação do serviço de captura de pacotes;
- Implementação do serviço de execução de comandos nativos;
- Implementação do serviço de rede neural artificial;
- Implementação do serviço de *log* com internacionalização;
- Implementação do serviço de envio de notificações por e-mail;
- Implementação do serviço de mensagens: mobilidade, solicitações, respostas e informativos;
- Implementação do serviço de localização: registro/remoção do DF (*Directory Facilitator*), pesquisa por containeres e agentes;
- Implementação do serviço de gerenciamento de linguagens e ontologias;

- Implementação do serviço de interface gráfica transportável.

IV) Criação de cenários para extensão e validação do protótipo:

- Especialização das classes básicas para cada problema proposto;
- Modelagem visual de uma ontologia incluindo os conceitos, ações, predicados e vocabulários necessários usando o software Protégé;
- Conversão do modelo ontológico para classes em Java;
- Validação e testes sobre os cenários implementados.

1.4 - ORGANIZAÇÃO DA DISSERTAÇÃO

O Capítulo 2 tem como objetivo fazer uma abordagem de segurança aplicada às redes de computadores. Inicialmente, são revisados os fundamentos de segurança computacional, alicerces para a implementação da proposta desse trabalho. Após isso, são discutidas as principais ameaças encontradas, a diversidade de ataques e as técnicas utilizadas para uma intrusão. Ao final do capítulo, são relacionadas algumas das principais técnicas e estratégias de segurança nos ambientes de redes, tais como o *firewall* e o sistema de detecção de intrusão (SDI).

O Capítulo 3 faz uma introdução dos conceitos relacionados aos agentes de software, levantando as respectivas definições, tipologias, linguagens de comunicação, formas de coordenação e, principalmente, a avaliação de sua aplicação na área de segurança computacional. Também é feito um estudo comparativo de algumas plataformas para suporte ao desenvolvimento e execução de agentes. Ao final é apresentado o padrão da FIPA (*Foundation for Intelligent Physical Agents)* que define as diretrizes para a interoperabilidade de agentes heterogêneos nos sistemas.

O Capítulo 4 consiste na parte principal desse trabalho, a proposta de um *framework* para o desenvolvimento de sistemas de detecção de intrusão baseados em grupos especializados de agentes.

O Capítulo 5 demonstra, de forma prática, a implementação de dois cenários para atuação dos agentes desenvolvidos com o *framework*: a detecção de usuários anômalos e a detecção de *port scanning*.

O Capítulo 6 conclui o trabalho e sugere algumas propostas futuras para a continuidade do desenvolvimento, tirando proveito da alta escalabilidade e manutenibilidade do modelo proposto.

O Apêndice A introduz o tema do surgimento da Internet e aborda resumidamente a arquitetura do protocolo TCP/IP. O principal objetivo dessa seção é servir como guia para os leitores que não estão familiarizados com o protocolo ou que desejam revisar seus conceitos, sendo que são necessários para a compreensão de boa parte desse trabalho.

O Apêndice B apresenta os diagramas das classes em UML desenvolvidas no protótipo do *framework.*

2 - VISÃO GERAL SOBRE SEGURANÇA NAS REDES DE COMPUTADORES

2.1 - CONSIDERAÇÕES INICIAIS

Segundo o dicionário da língua portuguesa [6], o termo seguro se refere a uma situação livre de riscos ou perigos, enquanto segurança, uma palavra derivada, significa um estado, qualidade ou condição de seguro. Em se tratando de sistemas computacionais, o termo seguro pode ser adequado para uma conotação de baixo índice de vulnerabilidades. Segundo o modelo de referência IS-74982 [7], uma vulnerabilidade é uma fraqueza que pode ser explorada para a violação dos sistemas ou das informações nele contidas. Logo, a segurança nas redes de computadores é o processo que visa uma maior proteção das informações e recursos das redes, por meio da redução das vulnerabilidades.

O maior desafio tem sido acompanhar o crescente investimento em novas tecnologias que objetivam, em primeiro plano, fornecer mais serviços e funcionalidades. Segundo Nakamura [8], existe uma relação inversamente proporcional entre o número de funcionalidades e a segurança nas redes, uma vez que a segurança poderá ser comprometida pelos seguintes fatores:

- Mais vulnerabilidades causadas pelo aumento no número de sistemas operacionais, aplicativos, protocolos e serviços;
- Maior complexidade no desenvolvimento e implementação de políticas de segurança;
- Maiores possibilidades de falhas na configuração de serviços e de sistemas de segurança;
- Ambientes cada vez mais propícios para a execução de ataques mais sofisticados.

Todavia, uma maior rigidez na segurança pode prejudicar a produtividade nesse ambiente, uma vez que será reduzido o número de funcionalidades do sistema, como por exemplo, o bloqueio de aplicações úteis como o FTP a fim de evitar a entrada de programas maliciosos. Outros exemplos típicos são os aplicativos de mensagens instantâneas que podem servir tanto como meio rápido de comunicação em empresas, quanto como meio de disseminação de vírus.

Fica claro que, mesmo com a aplicação de diversas medidas de segurança nas redes de computadores, é necessária uma constante evolução dessas técnicas para acompanhar o inevitável crescimento tecnológico, sendo, assim, a proteção total é impossível. O processo de manutenção da segurança deve alcançar uma boa relação entre as funcionalidades realmente necessárias e o nível aceitável de segurança.

Outro ponto em discussão é a necessidade de implantação de um conjunto de mecanismos de proteção em diversos níveis da corporação, pois os ataques possuem uma natureza diversificada. Ou seja, para que uma rede seja relativamente segura, se faz necessária a efetivação de um conjunto de medidas, como por exemplo, a instalação de um *firewall* para a proteção "de borda", de um sistema de detecção de intrusão para a monitoração interna e de normas de segurança para policiar os usuários, desse modo existirá um menor número de brechas.

Segundo pesquisa feita em 2005 pela CSI (*Compter Security Institute*) [12], essa diversidade de ataques tem causado grandes prejuízos, conforme mostrado na Figura 2.1. Como podem ser observadas, as maiores perdas não são relativas às invasões externas, mas sim por ataques internos, como a atuação de vírus, abusos e roubos por parte dos próprios funcionários. Esse paradigma coloca em cheque a estratégia de segurança apenas "de borda", já que os *firewalls* não podem tratar esses eventos internos. Como será visto, o presente trabalho atua justamente no escopo dos ataques internos, onde será proposta uma estrutura para a construção de ferramentas de monitoração e proteção contra essas ameaças.

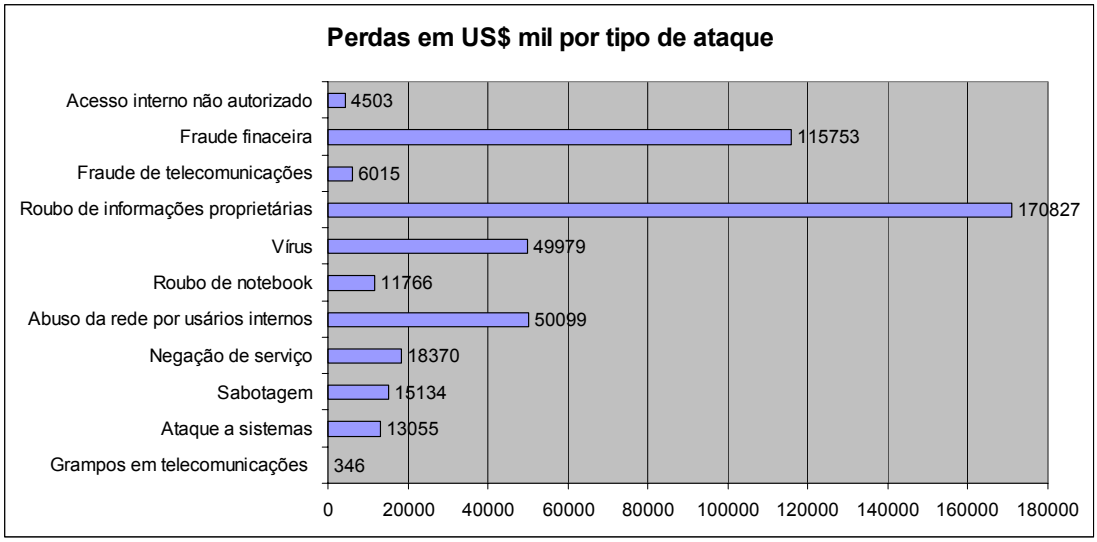


Figura 2.1 - Perdas em dólares por tipo de ataque (Fonte: CSI/FBI 2005)

Para uma melhor compreensão das questões relativas à segurança das redes baseadas nos protocolos do TCP/IP, bem como, das técnicas de ataques e mecanismos de defesa, o Apêndice A faz uma breve introdução desse popular protocolo que impulsionou em grande parte o surgimento da Internet. A leitura é recomendada para quem não está familiarizado com esse assunto.

2.2 - SERVIÇOS FUNDAMENTAIS DE SEGURANÇA

Stallings [9] define alguns serviços fundamentais de segurança para qualquer sistema computacional visando a proteção das informações transmitidas entre suas partes, ou seja, entre os recursos tecnológicos ou humanos que interagem com cada sistema:

- **Confidencialidade:** assegura que as informações que trafegam pela rede serão lidas apenas pelas partes autorizadas. Essa característica se estende a qualquer procedimento de leitura, impressão, e, até mesmo, em casos mais críticos, a simples revelação da existência desse objeto;

- **Integridade:** garante que apenas as partes autorizadas poderão modificar as informações transmitidas e que, em caso contrário, as informações permaneçam intactas. Qualquer operação como inclusões de trechos, remoções de trechos, exclusões, retardo, e encaminhamentos por entidades não autorizadas ferem os princípios da integridade;
- **Disponibilidade:** garante que os recursos computacionais estejam disponíveis quando solicitados pelas partes autorizadas após um período de tempo mínimo previamente conhecido;
- **Autenticação:** serviço mais abrangente que, de forma geral, se refere à correta identificação das partes do sistema, sejam elas: recursos tecnológicos, transações, usuários ou outros sistemas. Ou seja, é uma garantia de que a parte é quem ela diz ser;
- **Não-repúdio:** impõe que nenhuma parte autorizada poderá negar a manipulação ou a transmissão uma informação na rede;
- **Controle de acesso:** prevê que o acesso a um serviço da rede seja controlado por seu respectivo repositório.

2.3 - INTRUSOS, ATAQUES E AMEAÇAS

2.3.1. Potenciais intrusos

Segundo Anderson [10], podem ser identificadas três classes de intrusos:

- **Disfarçados:** intrusos externos não autorizados que penetram no sistema e adquirem contas válidas para a execução de suas ações;
- **Usuários válidos:** funcionários ou parceiros da própria corporação que extrapolam seus privilégios a fim de obter acesso às informações e aos recursos confidenciais;

- **Clandestinos:** intrusos internos ou externos que burlam o sistema de controle de acesso e de auditoria a fim de atuar livremente na rede.

Atualmente, qualquer causador de incidentes em sistemas computacionais é popularmente denominado como *hacker*. Diferentemente, a antiga definição se referia apenas aos indivíduos com alto conhecimento dos sistemas que usavam suas habilidades para executar invasões, sem o intuito de prejudicar as vítimas, mas sim como um meio de superar desafios e aumentar sua fama, ao contrário dos *crackers* que realmente visavam o prejuízo do alvo.

Algumas publicações sugerem uma categorização dos tipos de *hackers* segundo as próprias terminologias do meio, como mostrado na Tabela 2.1 [11]:

Tabela 2.1 - Principais nomes dos atacantes

Grupo	Descrição
Script kiddies ou newbies	São os *hackers* iniciantes. Não possuem experiência, nem suficiente conhecimento sobre os sistemas, mas estão sempre à procura de novas ferramentas e manuais de ataques, por isso, são potencialmente perigosos para as corporações, visto que na maioria das vezes desconhecem até mesmo as conseqüências dos seus atos. Dessa forma, causam bastante dor de cabeça para os administradores de rede que devem prever diversas situações e acompanhar periodicamente o lançamento de novas ferramentas de exploração de vulnerabilidades.
Cyberpunks	São os *hackers* mais experientes e extremantes anti-sociais, muitas vezes considerados como os *hackers* clássicos. São obcecados pela proteção de suas próprias informações e se dedicam às invasões cautelosas por puro desafio. Contribuem bastante na divulgação de novas vulnerabilidades em serviços, sistemas e protocolos.
Insiders	Funcionários ou contratados internos que invadem sua própria corporação. Segundo estudo do CSI (Computer Security Institute) [12], apesar de existir um maior quantitativo de ataques externos, os causadores de maiores prejuízos são os ataques internos, principalmente envolvendo o roubo de propriedade intelectual e divulgação para os concorrentes.
Coders	São os *hackers* que resolvem divulgar seus conhecimentos de segurança em bibliografias e palestras. Na maioria dos casos, após uma identificação e punição, resolvem mudar de lado, impulsionados pelo aspecto financeiro. Como exemplo, tem-se o clássico caso de Kevin Mitnick [13];

White hats	São considerados como os "*hackers* do bem". Vivem se atualizando e buscando soluções para novas vulnerabilidades em segurança. Trabalham de forma ética e profissional, atuando na maioria das vezes como consultores de segurança nas corporações. Muitas vezes precisam simular os ataques para a aplicação de suas correções, porém fazem isso em ambientes de laboratório próprios e controlados, sem colocar em risco seus clientes.
Black hats ou crackers	São os *hackers* que visam causar danos às vítimas, e muitas vezes são contratados por concorrentes que desejam alguma informação confidencial ou a falência da outra.
Gray hats	são *black hats* que procuram atuar como *white hats* por motivos de consultoria. Apesar de possuírem um vasto conhecimento sobre atividades de *hacking* não possuem uma formação disciplinada em segurança, por isso podem aproveitar o momento da consultoria para testar e divulgar novas vulnerabilidades daquela corporação. Muitas vezes são contratados após se revelarem como os autores de ataques recentes e detentores do conhecimento para as respectivas proteções.
Cyberterroristas	São os *hackers* que possuem uma motivação política ou religiosa para causar danos ou transmitir mensagens em escala mundial. Geralmente iniciam seus ataques por meio de *web defacements* ("pichações") para divulgar suas intenções e requisições.

2.3.2. Classificação geral dos ataques

Uma abordagem simplista para a classificação dos tipos de ataques seria apenas a distinção entre ataques ativos e passivos. Os ataques ativos executam alguma interação de criação, modificação ou eliminação dos fluxos de informações no sistema, enquanto que os ataques passivos apenas capturam as informações ou analisam o tráfego da rede sem inserir qualquer dado novo na comunicação.

Segundo [9], os ataques de forma geral podem ser divididos em quatro grandes categorias conforme apresentado na Figura 2.2:

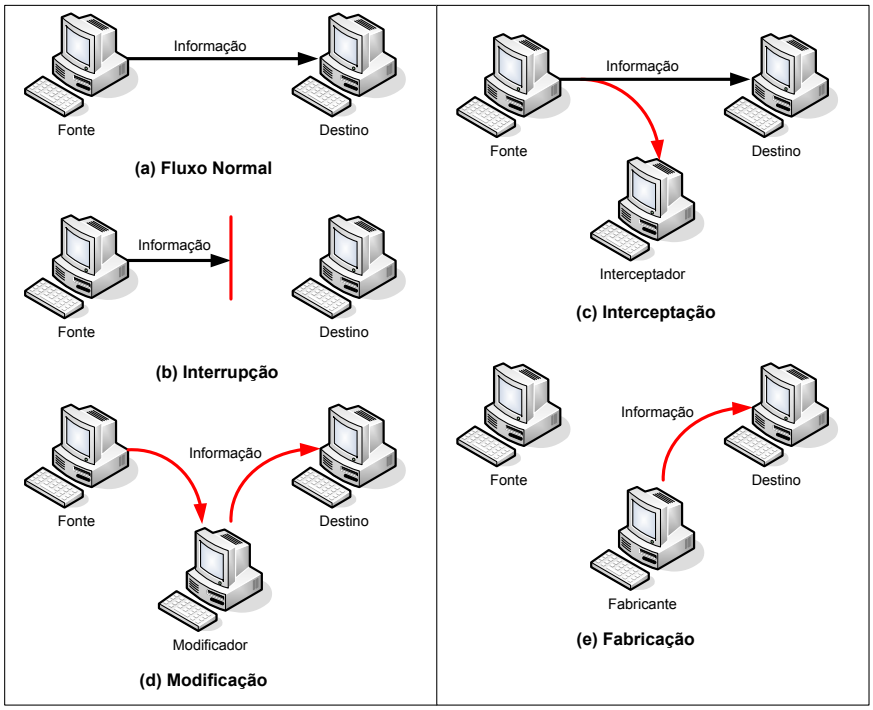


Figura 2.2 - Classificação dos ataques

- **Interrupção:** é um tipo de ataque com enfoque na disponibilidade de recursos do sistema. Por exemplo, os ataques de DoS (*Denial of Service*) que procuram retardar a resposta do sistema por meio de um bombardeio de pacotes inúteis. A interrupção se refere a um ataque ativo;
- **Interceptação:** uma parte não autorizada ganha acesso a um recurso. Esse ataque fere a confidencialidade da informação transmitida. A interceptação se refere a um ataque passivo;
- **Modificação:** uma parte não autorizada ganha acesso a um recurso e o utiliza para alterar o conteúdo das informações interceptadas. Esse ataque fere a integridade do sistema e é considerado como um ataque ativo;
- **Fabricação:** uma parte não autorizada gera informações como se fosse válida no sistema. Esse ataque fere a autenticidade da informação transmitida e, também, é considerado como um ataque ativo.

Quanto ao posicionamento do intruso em relação ao sistema alvo, os ataques ainda podem ser classificados como internos ou externos. Os ataques externos normalmente se referem aos ataques executados por *hackers* por meio da Internet.

Os ataques internos são realizados por indivíduos mal intencionados que atuam dentro do próprio ambiente corporativo. Geralmente são funcionários descontentes com sua situação e que se sentem subestimados em relação ao seu trabalho. Usam seus privilégios de acesso para roubar informações sem deixar rastros, pois conhecem bem a estrutura organizacional, operacional e cultural da instituição, por isso dificilmente são identificados. Também podem estar sendo subornados, enganados, e, até mesmo, ameaçados por interessados externos.

Como será visto mais adiante, a defesa contra os ataques internos é o foco desse trabalho, pois se trata do principal motivador para a construção de sistemas de detecção de intrusão, visto que apenas a segurança "de borda" exercida pelo *firewall* não contempla a proteção interna.

2.3.3. Técnicas de ataques de aquisição de informações

Em [8] são citadas algumas técnicas conhecidas de ataques que objetivam a obtenção de informações conforme a listagem apresentada na Tabela 2.2. Geralmente essas técnicas são aplicadas no preâmbulo de ataques mais sofisticados para um mapeamento inicial dos recursos ou obtenção de chaves de acesso aos sistemas. Por exemplo, para a realização de um ataque usando um *exploit* (explorador de *bugs*), o atacante deverá, no mínimo, conhecer quais serviços estão disponíveis e em quais versões, tarefa que pode ser realizada facilmente por uma ferramenta de varredura de portas (*port scanner)*.

Tabela 2.2 - Principais técnicas para aquisição de informações

Nome da Técnica	Descrição
Engenharia social	Exploração dos recursos humanos ao invés dos recursos tecnológicos para a obtenção das informações. O atacante procura ludibriar os usuários autorizados do sistema a fim de conseguir senhas e informações que lhe permitam invadir o sistema. Geralmente, ao exercerem a engenharia social, procuram simular uma falsa identidade.
Trashing	Técnica trivial, mas bastante utilizada que procura por informações descartadas nas lixeiras das corporações. Apesar de poder causar enormes danos às instituições, não é considerada ilegal.
Ataque físico	Técnica onde o intruso acessa diretamente o local do sistema, podendo, muitas vezes, furtar os recursos do sistema.
Consulta às informações públicas	Coleta de informações livremente disponíveis, principalmente por meio da Internet. Como exemplo tem-se, consultas em servidores de DNS, códigos fontes de páginas da *web*, entidades de registros de domínios, listas de discussões.
Sniffing	Captura de informações diretamente do fluxo de pacotes de rede, geralmente, após colocar a interface da máquina invasora em modo promíscuo.
Port scanning	Técnica que utiliza ferramentas para a descoberta dos serviços que estão ativos nos servidores da rede por meio do mapeamento de portas UDP e TCP.
***Scanning* de vulnerabilidades**	Técnica para execução de uma série de testes a procura por vulnerabilidades em serviços, protocolos, aplicativos e sistemas operacionais.
Firewalking	Técnica de exploração de redes similar a do `traceroute`, mas com recursos sofisticados que permitem o mapeamento de uma rede protegida por um *firewall*.
IP Spoofing	Técnica que simula um falso endereço na máquina do atacante a fim de esconder sua localização.
Criptoanálise	Técnica para decodificar informações protegidas pela criptografia. Geralmente busca o conhecimento da chave criptográfica ou do conteúdo das mensagens.

Algumas dessas técnicas, como o *port scanning*, serão utilizadas nos cenários propostos para avaliar o uso do *framework*, por isso serão mais bem estudadas e detalhadas nas próximas seções.

2.3.3.1. Considerações sobre o *port scanning*

Como já foi dito, a técnica de *port scanning* é bastante utilizada para obtenção de informações relativas aos recursos disponíveis da rede. Por meio dessa técnica, o atacante consegue realizar um mapeamento completo da rede em relação aos *hosts* e serviços disponíveis, dando um direcionamento sobre quais vulnerabilidades poderão ser exploradas. Algumas operações que são de interesse para os atacantes são:

- Descoberta dos h*osts* e *gateways* da rede e seus respectivos IPs;
- Identificação dos serviços acessíveis em cada host por meio do mapeamento de portas TCP e UDP;
- Identificação das versões de sistemas operacionais e serviços da rede.

De forma geral, a técnica consiste em forjar pacotes de conexão para uma lista especificada de IPs e portas ou para um *range* completo da rede. Com base na resposta daquela conexão, a ferramenta consegue determinar o estado do serviço envolvido, encerrando a conexão em seguida. A Figura 2.3 retrata alguns métodos utilizados para a realização do *port scanning* que estão presentes na maioria das ferramentas com essa finalidade, como por exemplo, no `nmap` [14], uma das ferramentas mais famosas.

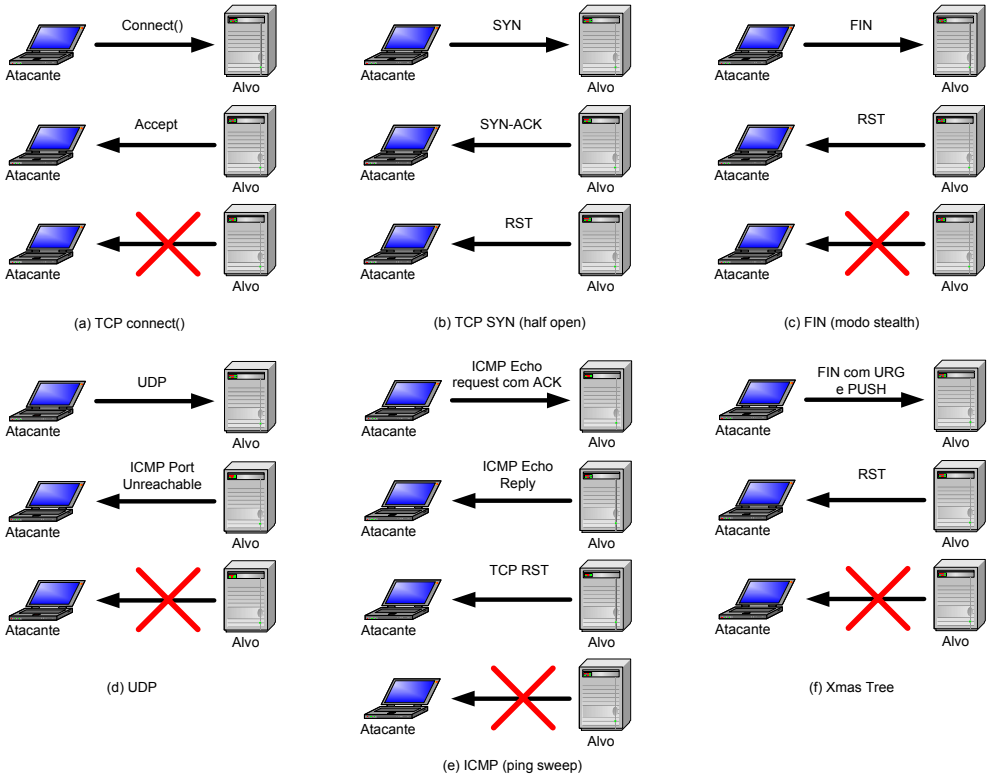


Figura 2.3 - Algumas técnicas de *port scanning*.

- **TCP connect():** forma mais rudimentar de varredura. A *system call* `connect()` é chamada apontando o endereço e porta do alvo. Caso a conexão seja bem sucedida, o respectivo serviço se encontra ativo. Esse método é facilmente localizado devido à abertura de conexões (Figura 2.3 a);
- **TCP SYN:** Faz uma abertura parcial da conexão TCP. Um pacote `SYN` é enviado para abertura de uma conexão no alvo. Caso seja retornado um pacote `SYN-ACK`, a porta de serviço está ativa, caso contrário, será retornado um pacote `RST`. Após receber o pacote de retorno do tipo `SYN-ACK`, será enviado um pacote `RST` para finalizar a conexão antes de sua abertura. Dessa forma não fica configurada uma conexão, sendo mais difícil sua detecção (Figura 2.3 b);
- **FIN:** Um pacote `FIN` é enviado para o alvo. Caso seja retornado um pacote `RST`, a porta está fechada, caso contrário, o pacote será ignorado apontando uma boa probabilidade da porta estar aberta. Esse método é mais complexo de ser detectado do que o método TCP SYN, pois a seqüência SYN/RST em curto

intervalo de tempo pode ser percebida por alguns *firewalls* mais específicos (Figura 2.3 c)

- **UDP:** Um pacote UDP de tamanho zero é enviado para cada porta do alvo. Caso seja recebida uma mensagem `ICMP port unreachable` será concluído que a porta está fechada, caso contrário será assumido que está aberta (Figura 2.3 d);
- **ICMP ou *ping sweep*:** Um pacote `ICMP echo request` juntamente com um `TCP ACK` é enviado para o alvo. Caso seja retornada uma mensagem `ICMP echo reply` então a porta esta aberta. Se for retornado um pacote `TCP RST`, significa que o alvo está ativo, mas não foi possível verificar o estado da porta. Se nada for retornado, então o alvo está desativo (Figura 2.3 e);
- **Xmas Tree:** Um pacote `FIN` com os *flags* `FIN`, `URG` e `PUSH` ligados é enviado ao alvo. Em caso de um retorno RST, a porta está fechada. Caso não seja recebida nenhuma resposta a porta poderá estar aberta;
- ***Null Scan:*** Mesmo método do Xmas Tree, porém com todos os flags derivados, dificultando um pouco a detecção.

Para dificultar a detecção das atividades de *port scanning*, são utilizados conjuntamente recursos que simulam uma atuação normal de uso da rede, driblando os *firewalls* e sistemas de detecção de intrusão. Dentre esses métodos se destacam:

- **Varredura randômica (*Random Port Scan*):** faz uma varredura aleatória nas portas de serviços. Dessa forma o padrão seqüencial não pode ser detectado;
- **Varredura lenta (*Slow scan*):** gera um retardo pré-estabelecido entre as varreduras para cada porta de serviço. Dessa forma o SDI acredita ser uma atividade normal;
- **Varredura fragmentada (*Fragmentation scanning*):** faz a fragmentação dos pacotes enviados na varredura. Com isso poderá confundir alguns métodos de detecção;
- ***Decoy*:** utiliza para a varredura uma série de endereços falsos, dando a impressão de que várias máquinas estão acessando normalmente aquele alvo. Além disso, a identificação de qual é a verdadeira origem dos pacotes fica comprometida;

- **Varredura coordenada (*Coordinated scans*):** distribui a tarefa de varredura para diferentes máquinas, onde cada uma será responsável por uma faixa de portas.

Apesar do *port scanning* também ser utilizado com boas intenções pelos administradores da rede, a técnica é considerada um ameaça pelos sistemas de detecção de intrusão, visto que, na maioria dos casos, é usada para a iniciação de um ataque. Além disso, estudos comprovam que devido à natureza truncada de alguns métodos de varredura, a própria pilha TCP/IP poder estar sendo comprometida, gerando, assim, falhas nos serviços [15]. Como exemplo, tem-se o clássico caso do travamento de estações Microsoft Windows 98 e a desabilitação do serviço `inetd` em alguns sistemas baseados em Unix após a aplicação de uma varredura do tipo TCP SYN.

2.3.3.2. Considerações sobre o *scanning* de vulnerabilidades

O *scanning* de vulnerabilidades pode ser considerado como uma técnica associada ao *port scanning*. Após o mapeamento dos *hosts* e serviços disponíveis na rede, os *scanners* de vulnerabilidades realizam diversos testes baseados nas versões dos serviços e sistemas especificados, a fim de determinar se existem riscos com base em um banco de assinaturas de vulnerabilidades conhecidas.

Segundo descrito em [16], um *scanner* de vulnerabilidades opera do seguinte modo:

a) Identifica quem está escutando em cada porta, atuando como um *port scanning*;

b) Identifica qual a versão do serviço que está rodando;

c) Pesquisa se há alguma vulnerabilidade conhecida associada com aquele serviço naquela versão;

d) Gera um relatório dos serviços vulneráveis naquele sistema.

Dentre as principais vulnerabilidades pesquisadas, em [8] são relatadas:

- Compartilhamentos de arquivos desprotegidos;
- Configurações superficiais ou incorretas;

- Softwares desatualizados ou versões com falhas de segurança;
- Senhas fracas;
- Condições para *buffer overflows* em serviços, aplicativos e sistemas operacionais;
- Aberturas para negação de serviço (DoS).

Do mesmo modo que ocorrem com os *port scanners*, os *scanners* de vulnerabilidades são ferramentas úteis para os administradores de segurança, mas perigosas nas mãos de um *hacker*. Por isso são consideradas ferramentas de ataque e devem ser detectadas sempre que possível.

2.3.4. Técnicas de ataques de negação de serviço

Os ataques de negação de serviço de DoS (*Deny of Service*) têm o objetivo de provocar a queda de serviços ou impedir que determinadas funções dos sistemas e aplicações consigam operar. Os ataques DoS costumam provocar grandes prejuízos devido à inabilidade de prover os serviços aos usuários durante o período de ataque, que pode levar horas e, até mesmo, dias, e ainda geram problemas subseqüentes como o comprometimento da integridade dos dados e da boa operação dos sistemas. Um dos exemplos mais famosos desse tipo de ataque foi o ocorrido em fevereiro de 2002 que ocasionou a paralisação dos *sites* da Yahoo! e da Amazon.com, ocasionando prejuízos na ordem de milhões de dólares [17].

- **Exploração de *bugs*:** Técnica que explora as brechas causadas por uma má implementação e concepção dos serviços, aplicativos e sistemas operacionais. Os maiores responsáveis por esse tipo de ataque são os próprios desenvolvedores de *software*, pois, muitas vezes, preferem priorizar as funcionalidades para depois se preocuparem com a segurança das mesmas. Caso freqüente é a exploração do *buffer overflow* (estouro de pilha) nos serviços que deixam margem para a execução de comandos no sistema remoto. O *hacker* explora *bugs* de implementação que comprometem a memória temporária para o armazenamento dos dados e deixa o sistema instável, causado sua indisponibilidade.

- **SYN *Flooding:*** Técnica de envio de um grande número de requisições de conexão (pacotes SYN) até que o servidor não consiga mais processá-las, causando a indisponibilidade do mesmo devido ao estoura da fila de conexões.
- **Fragmentação de pacotes de IP:** Técnica que aproveita o recurso de fragmentação e desfragmentação presentes em pacotes IP e definido pelo MTU (*Maximum Transfer Unit*). Como os pacotes são processados apenas após sua desfragmentação no destino, existe um empilhamento dos fragmentos antes de sua junção que deixa margem aos eventuais estouros em casos de instabilidades.
- ***Smurf e Fraggle:*** Técnica que envia um pacote `ICMP echo request(ping)` para o endereço de *broadcast* da rede utilizando um falo IP de origem (*IP Spoofing*) para a vítima do ataque. Como conseqüência todas as máquinas do domínio de *broadcast* irão responder à máquina atacada com um `ICMP echo reply` estourando a pilha TCP/IP da mesma. O *Fraggle* executa o mesmo procedimento, mas usando pacotes `UDP echo request`.

2.3.5. Técnicas de ataques de manipulação de pacotes

A manipulação de pacotes consiste em uma técnica bastante sofisticada onde determinados campos do cabeçalho TCP/IP são modificados ou mesmo pacotes inteiros são gerados a fim de simular uma conexão normal. Dessa forma o atacante consegue interagir com o sistema alvo como se fosse um usuário daquela conexão. Alguns exemplos desse tipo de técnica são:

- **Seqüestro de conexões *ou Hijacking:*** Técnica que executa o redirecionamento de conexões TCP para outra máquina, por meio da manipulação dos números seqüenciais (número do *acknowledgment*) presentes no cabeçalho do pacote. Para isso, o atacante altera os seqüenciais de forma que serão descartados no destinatário, e forja pacotes com os números válidos. Sendo assim o atacante consegue interceptar todo tráfego gerado daquela conexão, ataque conhecido também como *main-in-the-middle*.

- **Prognóstico de seqüencial do TCP:** Técnica de ataque onde pacotes são injetados na rede como se fossem originados por outras máquinas usando *IP Spoofing*. Para que isso seja possível, é necessário que haja a simulação dos números seqüenciais do TCP.

2.3.6. Técnicas de ataques em nível de aplicação

Os ataques em nível de aplicação são os mais comuns, pois se aproveitam das falhas que geralmente existem em qualquer serviço, protocolo e aplicativo desenvolvidos sem todos os cuidados de segurança, ou devido a grande complexidade ou por puro descuido. Comumente, são encontrados *scripts* na Internet*,* chamados de *exploits,* que exploram os *bugs* de uma dada versão de *software*.

Alguns exemplos de ataques em nível de aplicação são:

- ***Buffer overflow:*** Exploração de *bugs* de implementação que comprometem a estabilidade do sistema, causando a queda do serviço ou abrindo brechas para a execução remota de comandos nas vítimas;
- **Ataques na *web*:** Devido aos *bugs* existentes em servidores da *web*, principalmente quando possuem recursos de processamento de aplicações do lado do servidor (CGIs, scripts, container de aplicações), esses sistemas ficam à mercê de diversas formas de ataques. Dentre os mais comuns está o *Web Defacement* ou "pichação de site" onde o hacker aproveita as brechas para alterar as páginas inicias de sites corporativos geralmente deixando mensagens de desacato e depreciação para empresa vítima. Outros de importância são os ataques de *Web Spoofing* e *Hyperlink Spoofing*, onde o usuário é enganado com uma cópia do *site* ou *link* original a fim de obter dados confidenciais;

- **Injeção de comandos:** Outra técnica bastante comum que ameaça principalmente as aplicações baseadas em ambiente *web* é a injeção de comandos maliciosos na aplicação. Um exemplo clássico é a técnica de *SQL Injection* onde os campos de um formulário sem uma validação adequada são preenchidos com comandos de consulta no banco de dados, revelando muitas vezes dados confidenciais e obtendo listas de usuários do sistema. Isso ocorre por que determinados caracteres, por exemplo, os caracteres `"  '  "` (aspa simples) e `"/*"` (início de comentário), passados como parâmetros para o banco de dados truncam a rotina atual da SQL (linguagem de consulta ao banco) para a execução ou adição do comando especificado. Outra forma é passagem de parâmetros inválidos ou muito grandes por meio de requisições `HTTP` do tipo `POST` ou `GET`.

Os vírus, *worms* e cavalos de Tróia também são considerados ataques em nível de aplicação, porém devido ao alto poder de destruição e diversidade desses ataques serão tratados em uma seção à parte nesse capítulo.

2.3.7. Técnicas de ataques coordenados

Os ataques coordenados são considerados os mais sofisticados e de difícil defesa. Objetivam acima de tudo a indisponibilidade dos serviços, por isso são conhecidos também como ataques distribuídos de negação de serviço ou DDoS (*Distributed Denial of Service*).

Os ataques coordenados mais típicos usam um grande número de máquinas, popularmente chamadas de *zombies,* para fazer requisições em larga escala a um determinado alvo como se fossem requisições normais. Um ataque DoS comum poderia ser facilmente detectado visto que existiria apenas um endereço de origem com as conexões anômalas. Para preservar a localização do atacante, normalmente é acrescentado um nível adicional de máquinas, chamadas de *masters*, que controlam o parque de máquinas invasoras que também são vítimas do ataque, como mostrado na Figura 2.4.

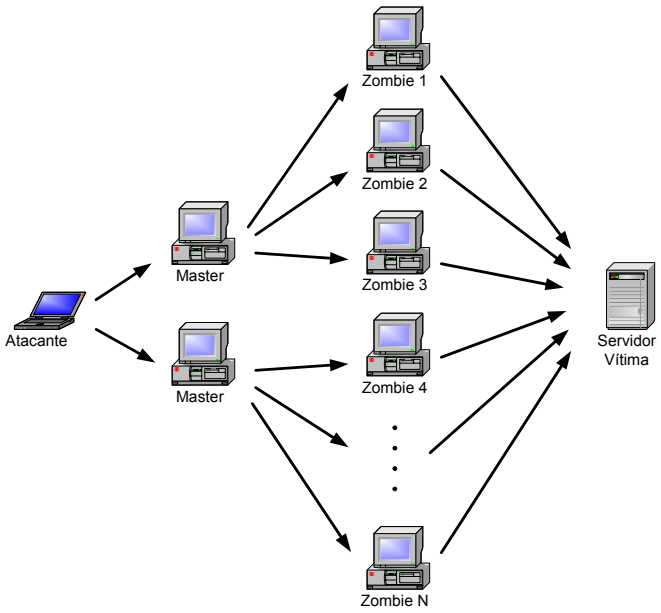


Figura 2.4 - Ataques coordenados (DDoS)

2.3.8. Técnicas de ataques para destruição em massa

2.3.8.1. Vírus e *Worms*

A maioria dos especialistas aceita a seguinte definição do termo vírus de computador: "um programa que se replica por meio da infecção de outros programas, logo, tais programas terão uma cópia do vírus" [18]. Ou seja, o objetivo primário de um vírus é a sua replicação por meio da infecção de outros programas.

A infecção é causada por um anexo adicionado às rotinas dos programas legítimos contendo o código malicioso. No momento em que esse programa é aberto o código anexado é executado.

O comportamento geral de um vírus pode ser estipulado do seguinte modo, segundo descrito em [16]:

a) O usuário chama um programa legítimo;

b) O código malicioso contido no programa é executado;

c) O código viral finaliza suas operações e devolve o controle para o programa legítimo.

Uma classificação sugerida para os vírus é mostrada abaixo:

- **Vírus de boot ou BSIs (Boot Sector Infectors) :** vírus exclusivos da arquitetura PC que infectam o MBR (*Master Boot Record*) ou o DBR (*DOS Boot Record*) por meio do acesso aos discos;
- **Parasitas:** vírus que infectam arquivos executáveis e se instalam na memória para a infecção de outros executáveis;
- **Híbridos:** vírus que se comportam tanto como vírus de setor de boot quanto como parasitas;
- **Polimórficos:** vírus que se alteram a cada infecção, dificultando a açã
- **Vírus de macro:** vírus que se instalam em documentos que suportem alguma linguagem de macro;
- **Vírus de script:** vírus que por meio do suporte às linguagens de *script* em navegadores web (*VBScript* e *JavaScript*) se instalam nas máquinas.

Os *worms* são um tipo especial de vírus que se proliferam massivamente por meio das redes. A diferença fundamental em relação aos vírus é que os *worms* não se replicam por meio da infecção em programas hospedeiros, mas, sim, por meio de sua própria cópia, logo, infectam o ambiente (sistemas operacionais e sistemas de e-mail) [18].

Uma classificação proposta para os tipos de *worms* é apresentada em [19]:

- ***Worms* de e-mail:** *worms* que utilizam os servidores e clientes de e-mail para se proliferarem;
- ***Worms* de protocolo:** *worms* que se proliferam por meio de protocolos não baseados em e-mail (IRC, FTP, TCP/IP Sockets).

Quanto à forma de transporte os *worms* ainda podem ser classificados como:

- ***Worms* de transporte automático:** *worms* que não necessitam de interação do usuário para se proliferarem, geralmente explorando bugs do sistema. Dentre os exemplos mais famosos estão: BubbleBoy, KAK e Code Red;
- ***Worms* de transporte via usuário:** worms que por meio da persuasão convencem os usuários a executarem anexos em e-mails com o código malicioso;
- ***Worms* híbridos:** *worms* que utilizam as duas técnicas de transporte. Como exemplos, têm-se o Nimda e as últimas versões do Code Red.

2.3.8.2. Cavalos de Tróia

Não existe uma única definição sobre o que é um cavalo de Tróia (do inglês, *trojan horse*), porém sua caracterização é bastante intuitiva. A RFC-1244 [20] sugere uma boa definição com base em seus comportamentos: *"um cavalo de Tróia pode ser considerado como um programa que faz alguma coisa útil, ou, ao menos, interessante. Porém, também faz coisas inesperadas, como roubar senhas ou copiar arquivos sem que o usuário tenha conhecimento".*

A história do cavalo de Tróia é contada em dois grandes poemas épicos associados à lendária guerra entre Tróia e a Grécia por volta de 1.200 a.C: "Odisséia", atribuída ao poeta grego Homero, escrita cerca de cindo séculos após os eventos, e "Eneida", do romano Virgílio, surgida oitocentos anos depois da época em que Homero teria vivido. Segundo eles, a Grécia declarou guerra à cidade de Tróia quando seu príncipe, Paris, seduziu e raptou Helena, considerada a mais bela mulher do mundo e, também, esposa de Menelau, rei de Esparta. Os gregos se empenharam na conquista de Tróia durante 10 anos, mas falharam em todas tentativas, principalmente, devido à grande dificuldade em abrir os resistentes portões do forte que protegia a cidade [21].

Tróia sucumbiu após trazer para dentro de sua cidade um enorme monumento em madeira, na forma de um cavalo, que foi deixado pelos gregos após a última derrota, acreditando ser um presente. No meio da noite, dezenas de soldados gregos, liderados por Ulisses, saíram de dentro do cavalo de madeira e abriram os portões da cidade para que o resto do exército entrasse. Foi o fim de Tróia [21].

Desse modo, o termo cavalo de Tróia é aplicado frequentemente para se referir aos programas que aparentemente são úteis e atrativos, mas guardam surpresas indesejáveis em seu conteúdo.

Os cavalos de Tróia apresentam três características fundamentais, como descritas em [16]:

- Existe algum tipo de funcionalidade útil ou atrativa nesses programas que justificam a sua execução;
- Executam comportamentos desconhecidos e não esperados por suas vítimas;
- Possui um conteúdo malicioso, ou seja, um trecho de código que foge do intuito aparente do programa.

2.4 - MECANISMOS DE PROTEÇÃO

Como foi visto cada tipo de ataque possui um objetivo específico. Alguns ataques procuram obter informações sigilosas, outros visam apenas o prejuízo do alvo. Diversos mecanismos de segurança podem atuar em conjunto de forma que combatam os diferentes tipos de ataques, inclusive, essa é a melhor abordagem para a gestão da segurança. Por exemplo, para ataques de aquisição de informações, poderiam ser cifrados e ainda poderia existir um esquema de autenticação no sistema que só permitiria o acesso de usuários autorizados.

Segundo pesquisa feita pela CSI em 2005 [12], as técnicas mais usadas se voltam para a autenticação, controle de acesso, sigilo das informações e monitoração das redes. Como visto na Figura 2.1, o recurso mais utilizado nas instituições é o Firewall (97%) que garante a proteção da rede interna contra ataques externos, fazendo o controle de acesso na borda da rede. Outro mecanismo bastante presente é o software antivírus (96%) juntamente com os sistemas de detecção de intrusão (72%) que tratam mais especificamente da segurança interna da rede.

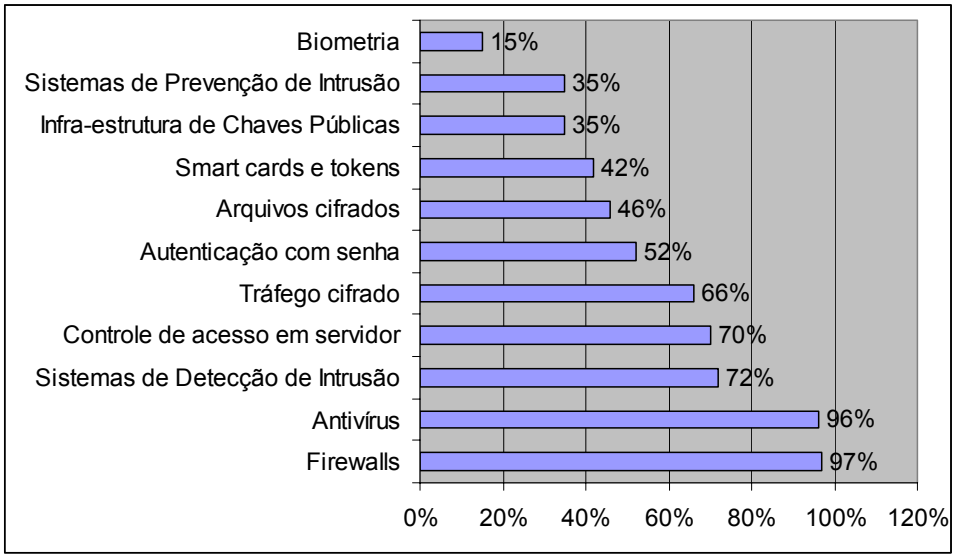


Figura 2.5 - Tecnologias de segurança mais utilizadas (Fonte: CSI 2005)

Outro mecanismo que vem sendo bastante aprimorado é o processo de autenticação de usuários. Tendo em vista o aumento da complexidade dos ambientes e do número de serviços e sistemas que necessitam da autenticação dos usuários, as soluções têm partido para um processo de autenticação em um único ponto, onde, uma vez autorizados, os usuários já possuirão as credenciais necessárias para acessar todos sistemas onde possuem permissão, sem a necessidade de novas solicitações de senhas. Esse processo costuma ser chamado de autenticação do tipo *Single Sign-On* (SSO) e pode ser associado a outros recursos como os dispositivos de biometria, certificados digitais, *tokens* e *smartcards*. Um exemplo dessa solução é o protocolo de autenticação Kerberos, que trabalha com *tickets* de validação residentes nas estações dos usuários após o primeiro *login*.

Nas próximas sessões serão apresentados alguns mecanismos de segurança bastante utilizados nas redes de computadores e será dada uma atenção especial aos de monitoração que se enquadram no tema desse trabalho.

2.4.1. Firewalls

O *firewall* com certeza é um dos componentes de segurança de redes mais importantes e conhecidos. Sua clássica definição foi dada por Steve Bellovin e Bill Cheswick [27] onde o *firewall* seria o ponto entre duas ou mais redes, no qual trafegam todos os pacotes. A partir desse único ponto, é possível controlar, registrar e autenticar todo tráfego da rede.

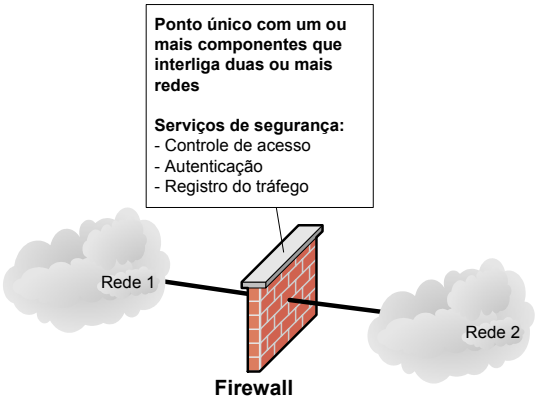


Figura 2.6 - Definição geral do firewall

Um *firewall* pode ser tratado como um conjunto de componentes, funcionalidades, arquiteturas e tecnologias utilizadas para o controle do acesso entre duas ou mais redes e, por meio dele, o acesso à rede interna pode ser restringido [16]. Geralmente, os dispositivos que implementam os *firewalls* podem ser computadores comuns, roteadores ou *appliances* (*hardwares* com um sistema operacional proprietário).

A boa atuação do *firewall* é consequência do modo como é configurado. Essa é uma tarefa que depende estritamente dos administradores da rede em conformidade com as políticas e normas de segurança definidas por cada organização. Existem duas abordagens para a configuração de um *firewall*. A primeira considera que: o que não é expressamente proibido é permitido, ou seja, o administrador da rede deve prever quais ações podem

infringir a política de segurança e preparar as regras de bloqueio dessas. A segunda considera que: o que não é expressamente permitido é proibido, sendo uma opção bem mais rígida e impactante para os usuários, porém mais segura.

2.4.1.1. Funcionalidades dos *firewalls*

Os *firewalls* são formados por diversos componentes, onde cada um tem uma funcionalidade específica para a segurança da rede. Em [28], Chapman define algumas das funcionalidades clássicas:

- **Filtros:** são os componentes responsáveis pela seleção de pacotes conforme as regras de filtragem definidas com base nos campos dos cabeçalhos. Podem aceitar ou descartar os pacotes;
- ***Proxies:*** atuam na camada de aplicação como *gateways* entre duas redes fazendo o reendereçamento das conexões de uma máquina interna para outra em uma rede externa. Por exemplo, para um usuário acessar uma servidor da Internet, ele primeiro se conecta à porta do *firewall* relativa àquele serviço, que, por sua vez, abrirá a conexão com o destino, reendereçando todo tráfego para o usuário;
- **Zona desmilitarizada:** conhecida como DMZ (DeMilitarized Zone) é uma rede intermediária entre a rede externa e a interna. Tem a função de prover os serviços externos da corporação por meio dos servidores do tipo *bastion host* como mostrado na Figura 2.7. Geralmente, vem protegida por dois *firewalls*: um "de borda" e um interno, impedindo, assim, que em caso de uma invasão a rede interna fique comprometida;
- ***Bastion hosts:*** são servidores que ficam em contato direto com a rede externa, por isso devem ter um cuidado especial em termos de segurança. Possuem apenas os serviços estritamente necessários a fim de evitar brechas no sistema. Geralmente qualquer servidor da DMZ deve ser tratado como um *bastion host*;

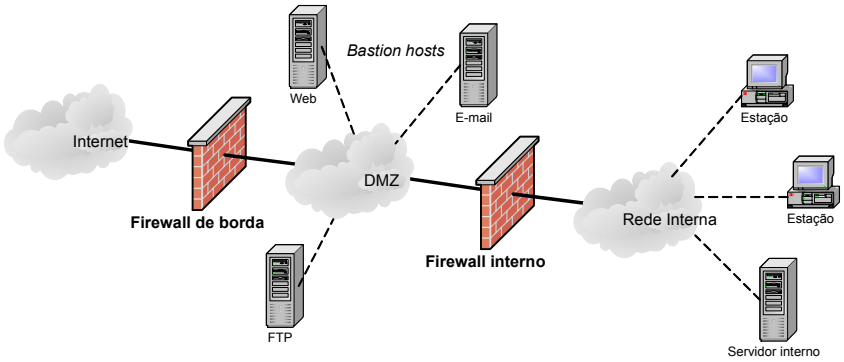


Figura 2.7 - Representação da zona desmilitarizada (DMZ) e serviços externos

- ***Network Address Translation:*** o recurso de NAT permite que uma rede interna endereçada com uma faixa de IP inválida [29] possa se comunicar com máquinas externas por meio da conversão e redirecionamento das conexões como se fossem com o endereço válido. É considerado um componente de segurança, pois esconde a configuração de endereços internos da rede;
- **Rede privativa virtual:** a VPN (*Virtual Private Network*) pode ser considerada como um enlace seguro entre redes corporativas, a fim de prover o sigilo, a integridade e a autenticidade da comunicação, mesmo, em meio a uma rede pública como a Internet. Para isso, utiliza de técnicas de criptografia;
- **Autenticação e certificação:** autenticação dos usuários com base nos endereços IP, senhas, certificados digitais, *tokens*, biometria ou outros;
- **Balanceamento de carga:** divisão do tráfego em dois ou mais *firewalls* visando um melhor desempenho, visto que um ponto único para o tráfego pode representar um gargalo na rede;
- **Alta disponibilidade:** adição de níveis de redundância nos componentes dos *firewalls* para que, em caso de contingência, o substituto entre em operação visando a continuidade da comunicação.

Recentemente, além dessas funcionalidades, outras tecnologias vêm convergindo para uma integração com os dispositivos de *firewall*, formando, assim soluções completas para a segurança "de borda" da rede, geralmente sobre a forma de *appliances*. Dentre as funcionalidades, as mais freqüentes são: autenticação dos usuários, criptografia do tráfego (VPN), qualidade de serviço (QoS), filtragem de conteúdo, filtragem de URL, filtragem de *spam* e antivírus corporativo.

2.4.1.2. Classificação dos *firewalls*

Em [8] é proposta uma classificação para os tipos de *firewall* de acordo com seu método de ação. Abaixo são enumeradas as principais tipologias, lembrando que a maioria das soluções recentes procura modelos híbridos que agregam todos os benefícios de cada tipo de *firewall*:

- **Filtros de pacotes:** opera na camada de rede e de transporte da pilha TCP/IP, fazendo a seleção de pacotes conforme regras pré-definidas envolvendo os respectivos campos do cabeçalho, como por exemplo, os endereços IP de origem e destino, as portas de origem e destino ou o sentido da conexão.
- **Filtros de pacotes baseados em estados:** assim como o filtro de pacotes comum, opera na camada de rede e de transporte da pilha TCP/IP, porém, além de considerar as regras envolvendo os campos do cabeçalho, também gerencia uma tabela de estados das conexões iniciais aceitas ou rejeitadas. Dessa forma, boa parte das ações do *firewall* se baseia no *cache* de conexões presente no *kernel*, acelerando o processo de busca e reduzindo a quantidade pacotes verificados;
- ***Proxies*:** esse tipo de *firewall* age como um servidor intermediário que faz o *relay* das conexões TCP, ou seja, o usuário se conecta a porta do *firewall,* que, por sua vez, abre outra conexão com o *host* remoto. A grande vantagem desse tipo de firewall é que não permite conexões diretas entre os hosts internos e externos;
- ***Firewalls* híbridos:** os *firewalls* híbridos mesclam as características dos filtros de pacotes e os *proxies*, usando as vantagens de cada tecnologia. Geralmente os serviços que exigem alto grau de segurança são tratados pelos *proxies* e os que exigem desempenho são tratados pelos filtros de pacotes;
- ***Firewalls* pessoais:** *firewall* que reside diretamente no host e protege somente as conexões endereçadas a ele. Recentemente essa tendência vem se tornando bastante comum, pois apenas a segurança em nível "de borda" da rede não é suficiente para tratar os modelos Peer-to-Peer (P2P) e redes privativas virtuais que exigem mecanismos individuais de filtragem de pacotes.

2.4.2. Sistemas de detecção de intrusão

Segundo [30], uma intrusão pode ser definida como: “um conjunto de ações que visam comprometer a integridade, confidencialidade ou disponibilidade de recursos”, sem a obrigatoriedade do sucesso das mesmas. Em [31], foi definido o termo detecção de intrusão, como: “o problema relacionado à identificação de ações de indivíduos que visam a utilização de um sistema computacional sem a autorização apropriada, ou ainda, a caracterização de abuso deliberado de privilégios por parte de usuários legítimos do sistema”.

Um sistema de detecção de intrusão (SDI) pode ser definido como um sistema computacional, geralmente composto por *hardware* e *software,* que visa executar a detecção da intrusão. Em melhores palavras, é o conjunto de mecanismos de segurança responsável pela detecção de atividades suspeitas ou impróprias na rede e pela notificação dos administradores sobre a forma de alarmes. Essa noção também deve ser estendida para a detecção de falhas decorrentes da má configuração ou mau funcionamento de sistemas e recursos de segurança que, de alguma forma, podem ameaçar a rede.

É importante ressaltar que os sistemas de detecção de intrusão foram concebidos para atender a necessidade de monitoração interna da rede, tarefa que não poderia ser tratada pelo *firewall*. Como foram introduzidos no início do capítulo, os maiores prejuízos são causados por ataques de natureza interna.

Frequentemente, os SDIs fazem uso de modelos estatísticos para a classificação de padrões no processo de detecção, por isso as avaliações são aproximadas e sempre contam com uma margem de erro. Um alarme errôneo de ataque é denominado como falso positivo, enquanto que a ausência de alarme na ocorrência de um ataque real é denominada de falso negativo. Segundo [32], todo sistema de detecção de intrusão gerará falsos positivos, principalmente porque, em alguns momentos, as atividades normais da rede se assemelharão bastante aos padrões de ataques. Os falsos negativos também sempre estarão presentes, visto que um SDI não conhecerá novos métodos de ataques. Sobretudo, os falsos negativos devem ser minimizados, pois são mais perigosos devido à falsa sensação de segurança.

2.4.2.1. Classificação

Após a avaliação de várias propostas para a classificação dos sistemas de detecção de intrusão, uma abordagem interessante foi a de organizar essas diferentes classes em termos das componentes do modelo genérico de arquitetura proposto por Bace [3], conforme mostrado na Figura 2.8.

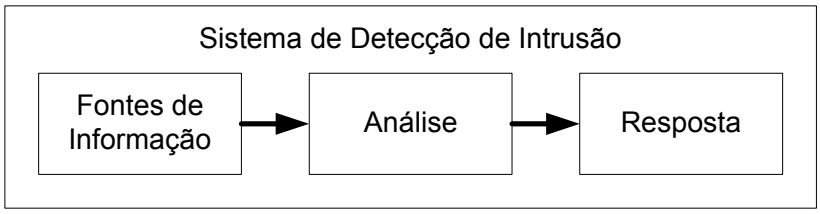


Figura 2.8 – Modelo genérico de arquitetura.

Segundo esse modelo, um SDI pode ser descrito em termos de três componentes:

- **Fontes de informação:** São os locais aonde os dados serão coletados pelo SDI. Podem ser obtidas em diferentes níveis do sistema, sendo que as mais comuns são referentes aos dados do tráfego da rede e dos sistemas executados nos *hosts*;
- **Análise:** É a principal parte do SDI, responsável por organizar e extrair o conhecimento das fontes de informação para a detecção das ocorrências de intrusões segundo uma metodologia de análise;
- **Resposta:** É o conjunto de ações tomadas pelo SDI após uma detecção de intrusão.

Dessa forma um SDI pode ser classificado quanto às fontes de informação, quanto à metodologia de análise (ou detecção) e quanto ao tipo de resposta. Em relação às fontes de informação, os SDIs podem se basear em dados capturados da rede, em dados existentes em cada *host* ou em ambos dados.

O sistema de detecção de intrusão baseado na rede (*Network-based Intrusion Detection System* - NIDS) monitora o tráfego em tempo real por meio de sensores posicionados na rede que possuem interfaces em modo "promíscuo", dessa forma todos os pacotes de cada segmento serão lidos. Em seguida, os conteúdos e cabeçalhos dos pacotes são analisados por uma console de gerenciamento central para que se detectem anomalias ou assinaturas de ataques.

Segundo [8] e [33], O NIDS traz as seguintes vantagens em relação aos sistemas *host-based*:

- Independente de plataforma;
- Os ataques podem ser identificados em tempo real;
- Baixo impacto no desempenho da rede;
- Poucos sensores bem posicionados podem monitorar toda rede;
- Difícil de serem descobertos devido à atuação passiva.

E as seguintes desvantagens:

- Perda de pacotes em redes de alta velocidade;
- Necessidade de alto processamento;
- Incapacidade de monitorar tráfego cifrado;
- Dificuldade na utilização em redes segmentadas;
- Pode ser driblado por técnicas de evasão, como: fragmentação, *slow scans*, uso de portas não convencionais e ataques coordenados;

O sistema de detecção de intrusão baseado no host (*Host-based Intrusion Detection System* - HIDS) faz o monitoramento do *host*, com base em informações dos registros do sistema ou de agentes de monitoração. Diversas informações podem ser coletadas e avaliadas, como o uso da CPU, processos de sistema, integridade dos arquivos, modificação de privilégios, *port scanning*, conexões de rede, *logs*, acesso aos arquivos e outros.

Segundo [8] e [33], O HIDS traz as seguintes vantagens em relação aos sistemas *network-based*:

- Atividades específicas do sistema podem ser analisadas detalhadamente, como o acesso a arquivos protegidos, modificação em permissões de arquivos, registros de *logon* e *logoff* do usuário;
- Ataques que ocorrem fisicamente no servidor podem ser detectados;
- Ataques que utilizam criptografia podem ser tratados já que há a decifragem das mensagens que chegam ao *host*;
- Opera independentemente da topologia de rede, não havendo dificuldades com redes segmentadas por *switches*;
- Não necessita de *hardware* adicional;
- Auxiliam na detecção de cavalos de Tróia e vírus, pois podem avaliar a integridade dos arquivos.

E as seguintes desvantagens:

- Apresenta problemas de escalabilidade devido à necessidade de configuração em cada *host*;
- É desenvolvido para cada plataforma específica, por isso é dependente do sistema operacional;
- Não é capaz de detectar ataques globais na rede;
- A invasão do HIDS pode comprometer as informações coletadas;
- Apresenta redução no desempenho do *host* monitorado.

Quanto à metodologia de análise dos SDIs, podem ser classificadas como baseadas em assinaturas ou em anomalias. A metodologia baseada em assinaturas, chamada de *Knowledge-based Intrusion Detection*, compara os dados coletados das com uma base de dados contendo assinaturas de ataques conhecidos. Como se baseia em informações conhecidas, essa técnica apresenta um baixo índice de alarmes falsos, porém exige que o SDI esteja em constante atualização.

A metodologia de detecção baseada em anomalia, chamada de *Behaviour-based Intrusion Detection*, analisa os desvios de comportamento normal dos usuários ou dos sistemas. Para isso são aplicadas técnicas de estatística e heurística a fim de detectar os padrões normais e anômalos. Um subtipo dessa metodologia é a análise de anomalia de protocolo onde são verificadas inconsistências nos fluxos dos pacotes tendo em mãos o padrão normal de formação de cada protocolo.

Quanto ao tipo de resposta, o SDI pode ser considerado de resposta ativa ou passiva. No caso da resposta ativa, são tomadas medidas automáticas quando ocorrem intrusões, como por exemplo, a ativação daquela conexão no *firewall*. Os SDIs de resposta passiva apenas relatam os alarmes deixando as intervenções à cargo dos administradores;

Uma ultima forma de classificação mais voltada para a arquitetura diz respeito ao posicionamento dos componentes do SDI. Nessa, podem ser classificados como monolíticos, quando o sistema inteiro reside em apenas um equipamento, ou como distribuídos, quando as partes dos sistemas residem em equipamentos distintos. Foram pesquisados alguns trabalhos já conhecidos que propõem uma abordagem monolítica como o IDES [34] e IDIOT [30]. Também, foram pesquisados modelos distribuídos, como o DIDS [34], GrIDS [35], EMERALD [36] e AAFID [37].

É importante observar que os projetos de pesquisa mais atuais e os novos produtos disponíveis no mercado vêm evoluindo no sentido de consolidar todas as alternativas vistas acima em uma única solução, surgindo, assim, os sistemas de detecção de intrusão híbridos. Dessa forma é possível aproveitar as vantagens de cada alternativa e tornar o SDI mais robusto. Por exemplo, pode-se monitorar tanto os hosts quanto o tráfego da rede, utilizar algoritmos de detecção que avaliam o comportamento do host e ainda consultam uma base de assinaturas de ataques, e, também, possibilitar que o sistema reaja em casos menos críticos e, em casos mais críticos, enviar o alerta para o administrador para que ele tome as medidas de proteção. A Figura 2.9 sumariza as principais classificações dos SDIs segundo a abordagem já introduzida.

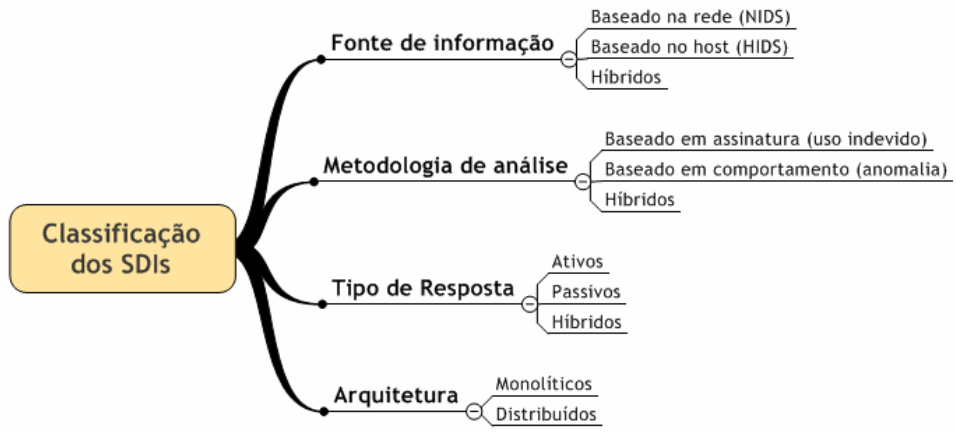


Figura 2.9 - Classificação dos SDIs

2.4.2.2. Características desejáveis

Algumas características desejáveis podem ser levantadas de forma que sejam genéricas o suficiente para que não dependam do tipo ou tecnologia empregada no sistema de detecção de intrusão. As seguintes características desejáveis que se aplicam a qualquer SDI foram apresentadas em [30]:

- Deve operar continuamente com mínima supervisão humana;
- Deve ser tolerante a falhas e fornecer degradação gradual do serviço, podendo recuperar-se em caso de panes acidentais ou decorrentes de ataques, procurando evitar ao máximo a paralisação completa;
- Deve ser resistente à subversão por meio de um auto-monitoramento para detecção de modificações maliciosas;
- Deve impor uma sobrecarga mínima para não impactar no desempenho geral da rede, ou seja, deve agir de forma transparente para os usuários;
- Deve ser flexível para suportar configurações específicas de cada ambiente;
- Deve ser adaptável às mudanças do ambiente e dos usuários no decorrer do tempo;
- Deve ser escalável de forma que acompanhe o crescimento do número de componentes da rede (estações, sistemas e usuários) sem perder qualidade nos resultados do processo de monitoração;

- Deve possibilitar uma reconfiguração dinâmica, permitindo que o administrador efetue mudanças em sua configuração sem a necessidade de reiniciar o sistema.

Outras características também são inerentes ao projeto de um SDI:

- Deve ter um bom nível de acurácia, proporcionando um baixo índice de falso-positivos e falso-negativos;
- Deve fazer a detecção em tempo-real, evitando que alertas tardios deixem comprometer o sistema;
- Deve possuir um bom tempo de resposta após a ocorrência de um alarme.

2.4.2.3. Vantagens de um SDI distribuído

Segundo [4], um SDI distribuído pode apresentar algumas vantagens que atendem às características desejáveis apontadas na seção anterior.

Primeiramente, em relação à tolerância a falhas e a degradação gradual de serviços, a natureza distribuída do sistema permite que parte dos seus módulos continue operando na decorrência de falhas por causas intencionais ou acidentais. Em contrapartida, um SDI monolítico possui um único ponto de operação, sendo que, ao haver falha nesse ponto, o sistema fica completamente comprometido.

Os SDIs distribuídos também são mais propícios à reconstrução dinâmica e adaptabilidade, pois pode haver a atualização ou adição individual de novos módulos mais especializados, sem a reinicialização do sistema como um todo.

3 - AGENTES DE SOFTWARE

Cada vez mais, o aumento da conectividade vem possibilitando o acesso a grandes volumes de informações eletrônicas que muitas vezes se encontram fisicamente distribuídas. Tirando proveito disso, as corporações estão ampliando seus negócios e fazendo grandes investimentos para aproximarem clientes, filiais, fornecedores e parceiros por meio das vias digitais.

Nesse cenário, é necessária a melhoria dos mecanismos de tratamento da informação para que as aplicações respondam satisfatoriamente. Sendo assim, os meios convencionais podem se tornar impróprios, promovendo a utilização de novas propostas como o processamento em paralelo e distribuído. Além disso, a complexidade para processar essas informações distribuídas exige que os processos tenham certa autonomia, mobilidade e inteligência, pois se torna inviável a operação desses sistemas apenas pela intervenção humana.

A tecnologia de agentes de software tem apoiado fortemente nesse intuito, oferecendo um novo paradigma para a solução de problemas complexos por meio de uma abordagem bastante flexível e apoiada no processamento distribuído. Além disso, o uso de agentes de software oferece vantagens em várias aplicações que necessitam de alta performance, alta disponibilidade, tolerância a falhas, adaptabilidade e escalabilidade, que é o caso dos sistemas de detecção de intrusão.

3.1 - DEFINIÇÕES

Existem várias propostas para a definição do que é um agente de software, porém não há um consenso geral devido à existência de diversos pontos de vistas dos mais variados campos da computação. Uma definição encontrada em [39] foi:

"[...] é uma entidade de software que funciona de maneira contínua e autônoma em determinado ambiente [...] possui a habilidade de conduzir as atividades de forma flexível, inteligente e sensível às mudanças nesse ambiente [...] em condições favoráveis, estaria capacitado a aprender a partir de suas experiências anteriores [...] é esperado que, ao habitar em um ambiente juntamente com outros agentes e processos, seja capaz de se comunicar com eles, e, talvez, de se deslocar de um local a outro para fazer isso [...]".

A FIPA (*Foundation for Intelligent Physical Agents*) já faz uma definição mais voltada para a implementação física do agente: "uma entidade que reside em um ambiente aonde interpreta dados por meio de sensores que refletem eventos do ambiente e executam ações que produzem efeitos no ambiente. Um agente pode ser software ou hardware puro [...]" [43].

De forma simples, um agente de software pode ser entendido como um programa autônomo que realiza tarefas continuamente e possui habilidades especiais que os diferenciam de programas comuns, por isso, é mais fácil compreender o conceito de agente de software em termos de dessas habilidades. Em uma abordagem de Inteligência Artificial, um agente ideal ainda seria capaz de adquirir o conhecimento necessário sobre seu ambiente para tomar decisões em determinadas situações.

A Tabela 3.1 resume as principais habilidades encontradas nos agentes de software segundo [40].

Tabela 3.1 - Principais habilidades dos agentes de software

Habilidade	**Descrição**
Autonomia	Controla suas próprias ações, podendo tomar decisões sem a intervenção contínua do usuário representado.
Reatividade	Reage adequadamente a ações externas no ambiente.
Pró-atividade	Toma iniciativa para realizar ações que levem ao cumprimento de seus objetivos.
Continuidade	Executa suas ações ininterruptamente.
Sociabilidade	Comunica-se com outros indivíduos (agentes, aplicações e humanos) por meio de uma linguagem de comunicação que o permite atuar colaborativamente ou competitivamente.
Inteligência	Muda seu comportamento com base na experiência anterior. Característica também referenciada como adaptabilidade ou aprendizagem.
Mobilidade	Move-se de um ponto ao outro desde que estejam conectados por uma infra-estrutura física de comunicação.
Flexibilidade	Suas ações não se resumem ao cumprimento de um roteiro pré-estabelecido.

Vale ressaltar que não é necessária a presença de todas essas propriedades em um único agente e que alguns autores consideram a autonomia, reatividade, pró-atividade e continuidade as características mais significativas nos agentes autônomos [39].

3.2 - TIPOLOGIAS DOS AGENTES

Essa seção visa apresentar uma visão geral das principais tipologias dos agentes e descrever as características que são mais acentuadas em cada uma. Boa parte das propostas de classificação apresentadas aqui foi baseada nos trabalhos propostos em [40], [33] e [42].

A Tabela 3.2 lista de forma arbitrária os diversos tipos de agentes de software, sem se preocupar com os diversos aspectos e dimensões possíveis. Essa lista pretende apenas dar uma visão geral das terminologias encontradas, pois muitos dos termos encontrados são redundantes ou sobrescrevem duas ou mais classes de agentes.

Tabela 3.2 - Tipologias dos agentes de software

Tipologia	Descrição
Agentes Inteligentes	Possuem um modelo interno simbólico de raciocínio que os permitem interagir, negociar e planejar ações com outros agentes para alcançarem um determinado objetivo. Nessa tipologia são evidenciadas as características de autonomia, aprendizagem e cooperação. São denominados também como agentes cognitivos ou deliberativos.
Agentes Colaborativos	Têm, em evidência, a habilidade de cooperação com outros agentes para executarem objetivos comuns. São denominados também como agentes cooperativos.
Agentes Competitivos	Ao invés de atuarem em conjunto com outros agentes por meio da colaboração, agem segundo suas próprias vontades, interagindo com outros agentes apenas por interesse em serviços concorrentes.
Agentes de Informação	Possuem o objetivo principal de localizar, manipular e ordenar as informações provenientes de várias fontes distribuídas de forma autônoma, tornando-as acessíveis para outros sistemas ou usuários.
Agentes de Interface	Atuam como interfaces intermediárias mais simples e robustas para os usuários que operam os sistemas. Trabalham por meio da delegação das ações dos usuários.
Agentes Estacionários	Não possuem qualquer habilidade de movimentação pela rede. Outras denominações encontradas foram: agentes estáticos ou situados.
Agentes Híbridos	Possuem mais de uma classificação possível. Baseiam-se na hipótese de que podem ser alcançados maiores benefícios com a utilização integrada das várias habilidades existentes.
Agentes Móveis	Podem se movimentar através da rede, interagindo com máquinas e usuários, coletando informações e retornando após alcançarem seus objetivos.
Agentes Reativos	Não possuem uma forma de representação simbólica interna de raciocínio e agem através de estímulos e respostas de acordo com o estado do seu ambiente.
Agentes Autônomos	São capacitados para tomar decisões e ações importantes para a conclusão de um objetivo sem a necessidade de intervenção humana. Podem agir independentemente em seu ambiente segundo suas próprias percepções.

Nesse trabalho será dado enfoque aos agentes autônomos e móveis por serem uma combinação indicada para os sistemas de detecção de intrusão visto que são necessários processos de monitoração e visitação contínua nos diversos servidores da rede. A forma de organização desses agentes em grupos também os caracterizará como agentes colaborativos.

3.3 - TÉCNICAS DE RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS

Atualmente, pode ser observado o amadurecimento de um novo campo dentro da Inteligência Artificial, a Inteligência Artificial Distribuída (IAD), que visa pesquisar e propor novos métodos para a resolução de problemas que têm como base as informações distribuídas. Nesse cenário, [38] afirma que a IAD assume uma metáfora baseada no *comportamento social*, onde um grupo distribuído de entidades procura uma solução colaborativa de problemas globais, ao contrário da Inteligência Artificial clássica que é baseada no *comportamento individual humano*.

Ainda em [38], é proposta uma divisão da Inteligência Artificial Distribuída em três abordagens, onde, em qualquer um dos casos, é utilizada a designação de agente para as entidades que participam nas atividades de solução dos problemas:

- ***Resolução Distribuída de Problemas (Distributed Problem Solving - DPS)*:** faz a decomposição de um problema complexo em módulos menores por meio de uma abordagem *top-down* orientada a um único objetivo específico, onde a maior parte da solução é inserida pelo próprio projetista;
- ***Sistemas Multiagentes (Multi-Agent Systems - MAS):*** se caracterizam pela existência de diversos agentes que interagem e colaboram entre si de forma autônoma para resolverem determinados problemas;
- ***Inteligência Artificial Paralela (Parallel Artificial Inteligence - PAI):*** se interessa mais pela performance na solução de problemas por meio da aplicação de diferentes algoritmos de computação paralela.

O objetivo dessa seção é mostrar as duas abordagens que mais se enquadram nos projetos de sistemas de detecção de intrusão baseados em agentes de software: a resolução distribuída de problemas (DPS) e os sistemas multiagentes (MAS). Originalmente, a proposta do *framework* possui uma abordagem aproximada com o MAS, principalmente devido as ações colaborativas e autônomas dos agentes que não visam necessariamente o processamento em paralelo, porém essa arquitetura também se aproxima de um DPS nas fases de designação dos grupos de agentes especializados. À medida que mais elementos de coordenação e elementos com maior flexibilidade forem sendo inseridos no sistema, haverá maior semelhança com a arquitetura dos sistemas multiagentes.

3.3.1. Resolução Distribuída de Problemas (DPS)

Na abordagem DPS, os agentes são designados para resolverem um problema em particular, dentro de uma concepção fechada de mundo. Isto significa que os agentes são projetados para resolver um tipo específico de problema apenas e não podem ser utilizados em qualquer outro problema, mesmo que seja similar. Desta maneira, o número de agentes será fixo, sendo que cada agente possui uma visão específica e incompleta do problema. Então, para a resolução de um problema, os agentes devem obrigatoriamente cooperar entre si, compartilhando conhecimento sobre o problema e sobre o processo de obter uma solução.

O projeto de um sistema DPS é realizado por um projetista que, primeiramente, realizará uma análise do problema a ser resolvido para, então, identificar os agentes necessários para a solução desse problema. Desta maneira, a tarefa de resolução será decomposta entre os vários agentes, buscando melhorar o processamento do sistema através da execução paralela. Esquematicamente, esta abordagem pode ser representada na Figura 3.1. [45].

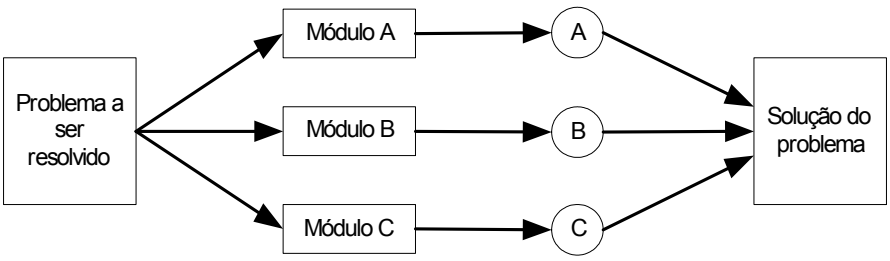


Figura 3.1 - Esquema de um projeto de DPS

3.3.2. Sistemas Multiagentes (MAS)

Pode-se dizer que os sistemas multiagentes são sistemas constituídos por vários agentes, ou seja, múltiplos agentes, que trabalham em conjunto para atingirem determinados objetivos. Dentro de um sistema multiagente, a definição dos tipos de agentes que serão utilizados dependerá de como o sistema tratará o problema, ou seja, podem ser utilizados agentes colaborativos, competitivos ou híbridos, etc.

Em um MAS, os agentes são projetados para resolverem qualquer tipo de problema, e não um problema em específico como acontece em DPS. Isso se deve, basicamente, porque, nos sistemas multiagentes, esses agentes são entidades autônomas que tem conhecimento da sua própria existência e da existência de outros agentes e, portanto, colaboram um com os outros para atingirem um objetivo comum dentro de um ambiente.

Desta maneira, na abordagem MAS, os agentes são primeiramente criados pelo projetista para após se estudar em que ambientes essa sociedade de agentes pode ser utilizada. O esquema de resolução de problemas na abordagem MAS pode ser observado na Figura 3.2.

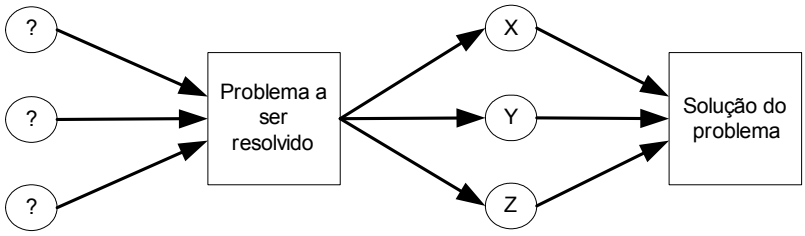


Figura 3.2 - Esquema de um projeto de MAS

Os agentes que pertencem a um sistema multiagente devem possuir algumas capacidades específicas para que as interações sejam possíveis. Primeiramente, os agentes devem saber de sua própria existência e da existência dos outros agentes, e devem possuir uma capacidade de comunicação por meio de uma linguagem comum. Além disso, cada agente deve possuir conhecimentos e habilidades para a execução de determinadas tarefas e, sempre que necessário, deve cooperar com outros agentes para atingir um objetivo global. Demazeau e Müller [44] e Sichman [45] colocam algumas considerações importantes para a abordagem MAS:

- Os agentes devem ser capazes de decomporem as tarefas baseados no conhecimento que eles possuem de si próprio e dos outros agentes;
- Como os agentes são autônomos, eles podem possuir metas próprias e decidirem o que fazer a qualquer momento;
- Os agentes possuem capacidade para resolverem seus problemas e os problemas que surgirem no ambiente;
- Os agentes podem entrar e sair do ambiente a qualquer momento. Portanto, em sistemas MAS os agentes devem ser capazes de modificar o conhecimento que possuem dos outros agentes do ambiente.
- Os agentes devem ser capazes de reconhecer modificações no ambiente quando estas ocorrerem, alterando sua representação interna do ambiente.

Em relação aos aspectos acima mencionados pode ser observado que as arquiteturas DPS, embora mais fácil de serem implementadas, são mais rígidas. Os sistemas MAS, por sua vez, apresentam maior flexibilidade, porém há um *overhead* devido a comunicação inter-agentes durante o processo de cooperação. Em [45] são relacionados os principais problemas encontrados na abordagem MAS:

- **Descrição, decomposição e alocação de tarefas:** como as tarefas mais complexas poderão ser descritas e decompostas em subtarefas mais específicas, e como essas subtarefas serão alocadas e, em que ordem deverão ser executadas;
- **Protocolo de Comunicação:** Que primitivas um protocolo de comunicação deveria utilizar num trabalho cooperativo;
- **Coerência, controle e coordenação:** como garantir um comportamento global coerente entre um conjunto de agentes, tendo cada um suas próprias habilidades e objetivos, e como deveria ser projetado o controle de tal sistema;
- **Conflitos e incertezas:** como podem ser resolvidos os conflitos que surgem, já que nenhum agente possui toda a informação do seu ambiente, e como dados incompletos podem ser distribuídos de forma garantir resultados coerentes;
- **Linguagem de programação e ambiente:** sob o ponto de vista computacional, quais as linguagens de programação que poderiam ser utilizadas em tais sistemas.

3.4 - COMUNICAÇÃO DOS AGENTES

A comunicação é fundamental para que haja a colaboração, negociação e coordenação entre os agentes. Essa comunicação deve ser bem definida para que os objetivos desses sistemas sejam alcançados de forma eficiente.

Segundo [69], existem diversas estratégias de comunicação entre os agentes. Dentre elas, podem ser citadas:

- **Comunicação direta:** Cada agente se comunica diretamente com qualquer outro agente, sem intermediários, como mostrado na Figura 3.3 (a). Nessa estratégia cada agente precisa saber da existência do outro e de como endereçar as mensagens para ele. A principal vantagem é a de não existir um gargalo na comunicação devido ao uso de um agente intermediário, porém, em contrapartida, pode tornar a implementação muito complexa e pouco escalável.
- **Comunicação por sistema federado:** Os agentes fazem uso de algum sistema ou mesmo de um agente para coordenar suas atividades de comunicação. Esses agentes especiais são chamados de facilitadores ou mediadores, como mostrado na Figura 3.3 (b). Essa estratégia incentiva o crescimento do número de agentes, já que existe um maior controle da comunicação.
- **Comunicação por difusão de mensagens (*broadcast*):** O agente envia sua mensagem para todos os agentes, independente se essa mensagem será útil ou não para determinados agentes, como mostrado na Figura 3.3 (c). Essa estratégia é utilizada quando se deseja enviar mensagens para todos os agentes simultaneamente ou para quando não se conhece o endereço de determinado agente.
- **Comunicação por quadro negro (*backboard*):** É o mesmo modelo de memória compartilhada utilizado na Inteligência Artificial. Os agentes fazem uso de um repositório comum para buscar e gravar as mensagens, conforme a Figura 3.3 (d).

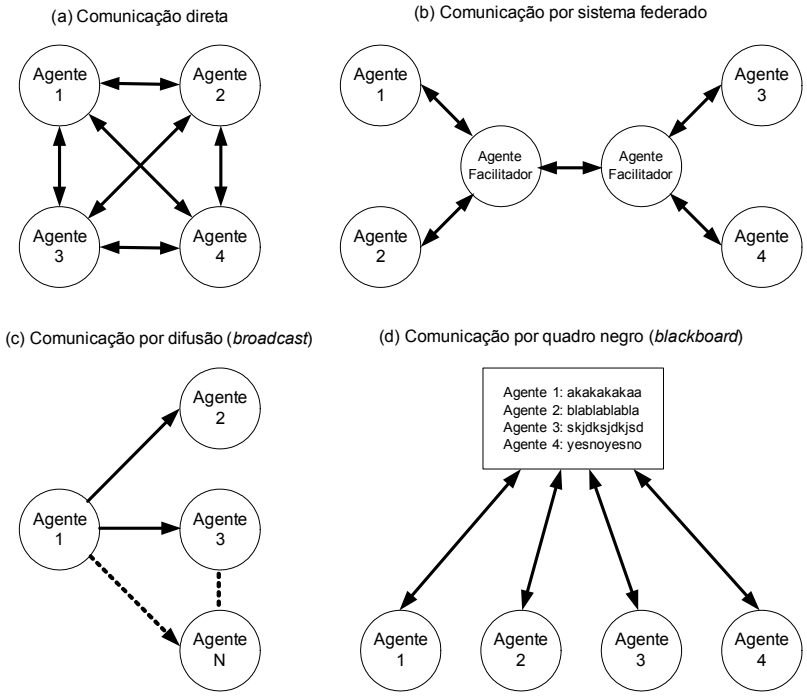


Figura 3.3 - Estratégias de comunicação entre agentes

3.4.1. Ontologia

Uma das definições mais conhecidas para ontologias é apresentada por Gruber em [70]: "Uma ontologia é uma especificação explícita de uma conceitualização.[...] Em tal ontologia,definições associam nomes de entidades no universo do discurso (por exemplo, classes, relações, funções etc. com textos que descrevem o que os nomes significam e os axiomas formais que restringem a interpretação e o uso desses termos) [...]." O termo *conceitualização* se refere a um conjunto de objetos, conceitos e outras entidades que existem em um domínio e os relacionamentos entre eles [71]. Uma conceitualização é uma visão abstrata e simplificada do mundo que se deseja representar.

Em [73], Borst apresenta uma definição mais prática, que será adotada neste trabalho: "Uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada". O termo formal significa que pode ser interpretada por computador, o caráter explicito se deve ao fato de que os conceitos utilizados e suas restrições são prévia e explicitamente definidos, e, finalmente, o termo compartilhado se refere a um conhecimento consensual, compreendido por mais de um indivíduo e aceito por um grupo.

A utilização de ontologias permite a definição de um alicerce para integração em nível de conhecimento de sistemas inteligentes, incluindo os sistemas baseados em agentes inteligentes, trazendo os seguintes benefícios [75]:

- **Colaboração:** possibilitam o compartilhamento do conhecimento entre os membros interdisciplinares de uma equipe. Esse compartilhamento também se aplica aos grupos de agentes inteligentes;
- **Interoperabilidade:** facilitam a integração da informação, especialmente em aplicações distribuídas, como por exemplo a dos agentes;
- **Informação:** podem ser usadas como fonte de consulta e de referência do domínio;
- **Modelagem:** as ontologias são representadas por blocos estruturados que podem ser reusáveis na modelagem de sistemas no nível de conhecimento;
- **Busca baseada em ontologia:** recuperar recursos desejados em bases de informação estruturadas por meio de ontologias. Desta forma, a busca torna-se mais precisa e mais rápida, pois quando não é encontrada uma resposta exata à consulta, a estrutura semântica da ontologia possibilita, ao sistema, retornar respostas próximas à especificação da consulta.

Em suma, uma ontologia é uma especificação explícita de objetos, conceitos e outras entidades que são assumidas como verdadeiras e uma área de interesse, além das relações entre esses conceitos e restrições expressados por meio de axiomas.

No contexto desse trabalho, escolheu-se a utilização de ontologias para a definição da linguagem de comunicação dos agentes, tendo em vista que sua principal característica está centrada no reuso e compartilhamento. Dessa forma, o vocabulário pode ser definido, sem ambigüidades, com grande estruturação e organização semântica.

3.4.2. Linguagem de comunicação

A linguagem de comunicação pode ser entendida como o instrumento que permite a comunicação por meio do compartilhamento do conhecimento de um dado domínio. No caso dos agentes, ela consiste em uma representação simbólica que expressa seus desejos, crenças, intenções, etc. A linguagem de comunicação dos agentes (ACL – *Agent Communication Language*) provê as primitivas necessárias para que o agente implemente essa representação simbólica.

Como modelo de comunicação de agentes em uma linguagem ACL é usado a própria comunicação humana juntamente com a teoria de atos de fala (*Speech Act Teory*) [76]. Nessa teoria as mensagens estão associadas a atos performativos que representam a vontade do agente sobre a informação contida na mensagem. [77] lista algumas características dessa teoria:

- Derivada da análise lingüística da comunicação humana;
- Com uma linguagem, o individuo da conversação não só efetua uma declaração, como também realiza uma ação;
- Mensagens são ações ou atos comunicativos.

McCarthy em [78] ressalta que a teoria dos atos comunicativos relaciona: expressões humanas (ou de máquinas) em diferentes categorias dependendo da intenção do falador (ser humano ou agente); o efeito sobre o ouvinte (ser humano ou agente) e qualquer outra manifestação física que implique na expressão de uma ação. Conforme descritas em [79], essas categorias são mostradas na Tabela 3.3.

Tabela 3.3 - Categorias dos atos comunicativos

Categoria	**Intenção**	**Exemplo**
Representativas	Expressa o contexto da proposição	"Em relação à segurança da rede.."
Diretivas	Solicitação ou comando	"Agente de coleta, mova-se para..."
Comissivas	Promessa ou ameaça	"Se não fizer isso, terá que ... "
Expressivas	Agradecimento e desculpas	"Obrigado pela ajuda..."
Declarativas	Relativas a mudanças no ambiente	"Agora, eu sou o coordenador"
Veredictativas	Julgamento passado	"Foram achados pacotes anômalos"

3.5 - FIPA

Foundation for Intelligent Phisical Agents ou apenas FIPA é uma organização sem fins lucrativos que tem por objetivo propor padrões de interoperabilidade de agentes heterogêneos e interativos e de sistemas baseados em agentes. Foi fundada em 1996 e possui sede em Genebra.

Oficialmente sua missão é descrita da seguinte forma: "A promoção de tecnologias e especificações de interoperabilidade que facilitem a comunicação entre sistemas de agentes inteligentes no contexto comercial e industrial moderno." [43]. Resumindo, propõe a interoperabilidade entre agentes inteligentes.

3.5.1. Modelo de referência

O modelo de referência definido pela FIPA para plataformas de agentes pode ser observado na Figura 3.4.

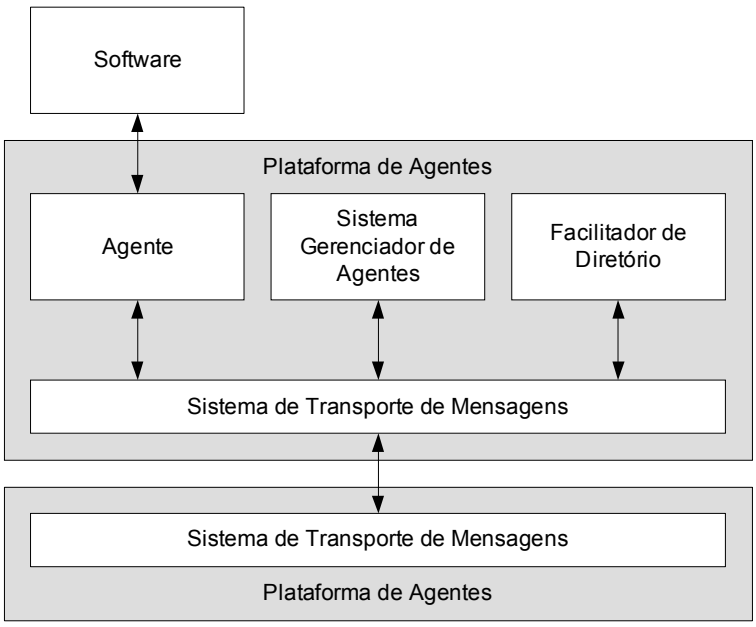


Figura 3.4 - Modelo de referência FIPA

De acordo com [43], o modelo de referência é composto pelos seguintes componentes:

- **Software:** Representa todas as entidades externas de software que não são consideradas agentes. Incluem-se nesse âmbito as ferramentas corporativas, banco de dados, componentes remotos, dentre outros.
- **Agente:** É o agente propriamente dito que procura realizar as tarefas de acordo com o objetivo da aplicação. Reside na própria plataforma de agentes e pode se relacionar tanto com a aplicação externa quanto com outros agentes por meio de troca de mensagens.
- **Sistema Gerenciador de Agentes (AMS - *Agent Management System*)**: É o componente que gerencia o acesso e uso da plataforma de agentes. É o responsável pelo controle do ciclo de vida dos agentes e pela manutenção de um diretório de identificadores (*AID – Agent Identifier*) e estados dos agentes.
- **Facilitador de Diretório (DF - *Directory Facilitator*):** Provê o serviço de páginas amarelas. Os agentes podem se registrar nesse local informando seu identificador e os serviços prestados.
- **Sistema de Transporte de Mensagens (ACC - *Agent Communication Channel*):** Fornece todos os serviços de comunicação entre os agentes dentro ou fora da plataforma.
- **Plataforma de Agentes:** Ambiente aonde os agentes podem ser desenvolvidos e executados. Geralmente é composta por uma infra-estrutura física distribuída ou não, sistemas operacionais, aplicações, componentes de gerência de agentes e até mesmo outros agentes prestadores de serviços.

A FIPA não visa fazer especificações das estruturas internas dos agentes, pois estas dependem de cada aplicação. Todavia, procura garantir a interoperabilidade entre os agentes de diferentes plataformas por meio da especificação dos comportamentos externos e interfaces dos agentes, como por exemplo, definindo os padrões dos protocolos, linguagens de conteúdo, ACL e outros [43].

3.5.2. Ciclo de vida do agente

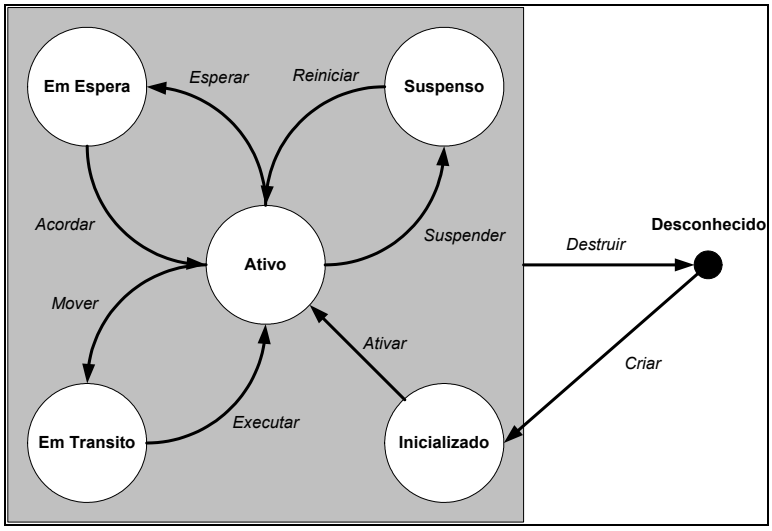


Figura 3.5 - Estados possíveis do agente

A definição de cada estado é explicitada abaixo conforme a Figura 3.5:

- **Iniciado:** o agente foi criado, mas ainda não se registrou no MAS. Dessa forma não tem nome ou endereço e não pode se comunicar com outros agentes.
- **Ativo:** o agente está registrado no MAS, possuindo nome, endereço e capacidade de comunicação.
- **Suspenso:** o agente está paralisado, ou seja, nenhum processamento está sendo executado.
- **Em espera:** o agente está bloqueado, esperando por algum evento. Quando o evento aguardado ocorrer, irá acordar e voltar ao estado ativo.
- **Em transito:** o agente está se movimentando para outro local. Todas as mensagens emitidas a esse agente nesse instante serão armazenadas em um *buffer* para, após a chegada no novo local, serem encaminhadas.

3.5.3. FIPA-ACL

A FIPA-ACL é a linguagem de comunicação de agentes desenvolvida pela FIPA que se baseia também em performativas derivadas da "teoria dos atos da fala". Ela define as performativas organizadas em 4 categorias: transferência de informação, negociação, ação e gerenciamento de erros [43].

Um exemplo de comunicação utilizando a linguagem FIPA-ACL pode ser visto na, que exemplifica uma troca de informações entre dois agentes.

Tabela 3.4 - Exemplo de mensagem FIPA-ACL

```
(inform
: sender Agente-A
: receiver Agente-B
: content 'status=OK'
: language fipa-sl
: ontology security
)
```

Nesse exemplo, um agente identificado como Agente-A enviou uma mensagem informativa (performativa *inform*) a Agente-B. A linguagem da mensagem é a FIPA-SL e o seu conteúdo está relacionado a uma certa informação do domínio de segurança (*security*) O conteúdo da mensagem apenas contém o texto “coleta de pacotes da rede”.

A FIPA-ACL disponibiliza um conjunto limitado e bem definido de atos comunicativos, porém suficiente para representar qualquer situação que possa surgir na comunicação entre agentes. A seguir, essas performativas são descritas de forma informal:

Tabela 3.5 - Atos comunicativos da FIPA-ACL

Ato	Descrição
accept-proposal	Declaração de aceite de uma proposta feita por outro agente.
agree	Aceite do agente para atender a uma requisição anterior feita com a performativa *request*.
cancel	Indicação que o agente não deseja mais executar determinada ação solicitada.
cfp	Indica interesse em um certo serviço e espera as propostas dos agentes que podem realizá-lo (*call for proposals*)
inform	Mecanismo básico para a comunicação de informações. O agente emissor deseja que o receptor acredite em seu conteúdo.
not-understood	Permite que o agente informe que não entendeu a razão de uma ação executada por outro agente.
propose	Permite que um agente faça uma proposta para outro agente, por exemplo, em resposta a uma mensagem *cfp* anteriormente enviada.
refuse	Usada por um agente para informar a outro que não executará determinada ação.
reject-proposal	Usada por um agente para indicar a outro que não aceita uma proposta feita durante uma negociação.
request	Permite que um agente peça a outro que execute uma ação.

3.6 - JADE

JADE (*Java Agent DEvelopment framework*) é uma plataforma para o desenvolvimento e execução de sistemas baseados em agentes de software que segue as recomendações da FIPA. É escrito totalmente em Java, por questões de portabilidade, e distribuído sob a licença de software livre LGPL (*Lesser General Public License*).

Segundo o site oficial do projeto [55], o principal objetivo da JADE é simplificar o desenvolvimento de sistemas multiagentes garantindo a interoperabilidade com outros sistemas por meio de um abrangente conjunto de serviços padronizados que facilitam a comunicação entre agentes de acordo com as especificações da FIPA.

3.6.1. Características

A JADE fornece uma estrutura robusta para o desenvolvimento sistemas multiagentes, possuindo as seguintes características:

- **Plataforma distribuída de agentes:** Pode ser dividida em vários servidores que serão interligados por meio do protocolo RMI (*Remote Method Invocation*). Para isso, em cada máquina deve ser iniciado um repositório de agentes (container) sobre a JVM aonde residirão os agentes;
- **Interface gráfica de gerenciamento:** Interface gráfica baseada na biblioteca Swing que possibilita a visualização dos containeres e respectivos agentes, além de possibilitar a gestão do ciclo de vida de cada agente;
- **Ferramentas de depuração:** Ferramentas que auxiliam na depuração de todos os processos e no acompanhamento da comunicação entre agentes.
- **Suporte ao processamento concorrente (*multithreading*)**: Introdução de um modelo mais natural baseado em comportamentos (*Behaviours*) dos agentes.
- **Complacência com a FIPA**: Presença dos serviços especificados pelo modelo de referência da FIPA: o sistema gerenciador de agentes (*MAS – Agent Management System*), o facilitador de diretório (*DF – Directory Facilitator*) e o canal de comunicação (*ACC – Agent Communication Channel*). No JADE, esses serviços são caracterizados também por agentes de software.
- **Transporte de mensagens:** Serviço de transporte de mensagens seguindo o padrão FIPA-ACL.
- **Automação de registros -** Registro e remoção automática dos agentes do AMS quando os mesmos forem iniciados ou finalizados.
- **Serviço de nomes (*Naming Service*) –** Quando os agentes são iniciados recebem um identificador chamado de GUID (*Globally Unique Identifier*) que garantem a unicidade em toda plataforma, ou seja, em todos os containeres.
- **Biblioteca de protocolos de iteração FIPA –** Dispõe de um conjunto de protocolos de iteração da FIPA prontos para a utilização.

3.6.2. Arquitetura da plataforma

Segundo descrito em [55], a arquitetura da plataforma JADE incentiva a criação de diversos containeres que podem ser distribuídos de forma arbitrária entre um ou mais servidores como exemplificado na Figura 3.6 (Servidor 1, Servidor 2 e Servidor 3).

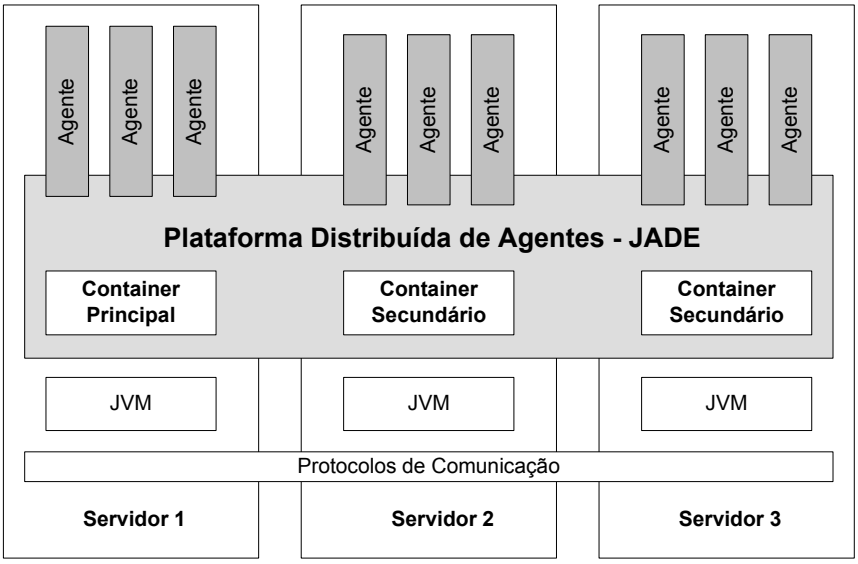


Figura 3.6 - Arquitetura da plataforma JADE

Os containeres de cada servidor atuam sobre uma JVM (*Java Virtual Machine*) que garante a independência de plataforma e permite que seja implementado um método de comunicação padrão entre os agentes, mesmo que estejam em ambientes heterogêneos.

Como pode ser observado, sobre cada JVM é criado um container do JADE que fornece todos os serviços já mencionados para a instalação e execução de um ou mais agentes. Sempre deverá ser designado um container principal (*Main-container*), aonde residirão os agentes de serviço da plataforma (AMS, ACC e DF) de forma que os outros containeres possam se integrar a ele. A plataforma como um todo será composta pelos diversos containeres, formando um ambiente único de execução.

Para otimizar o transporte de mensagens entre os agentes em cada cenário, a JADE trabalha com diferentes métodos de comunicação, como apresentado na Figura 3.7.

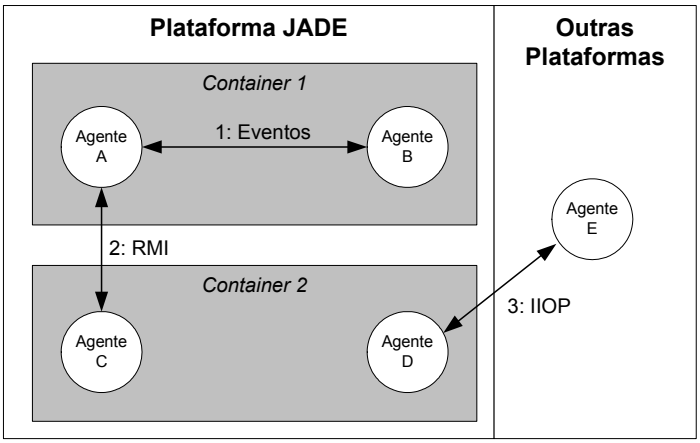


Figura 3.7 - Métodos de comunicação da plataforma JADE

Se a comunicação é feita entre agentes do mesmo container (Agente A / Agente B), é utilizado o próprio recurso de geração e tratamento de eventos do Java. No caso da comunicação entre agentes distribuídos em uma mesma plataforma (Agente A / Agente C), é utilizado o protocolo RMI (*Remote Method Invocation*) que disponibiliza um conjunto de classes e interfaces que encapsulam diversos mecanismos de troca de dados, a fim de simplificar a execução de chamadas de métodos remotos em diferentes espaços de endereçamento. Por fim, para garantir a comunicação com outras plataformas de agentes (Agente D / Agente E), é utilizado o protocolo IIOP (*Internet Inter-ORB Protocol*) da especificação de interoperabilidade CORBA (*Common Object Request Broker Architecture*). De acordo com [56], o IIOP permitiu que o CORBA se tornasse uma solução definitiva para a interoperabilidade entre objetos que não estão atrelados a uma plataforma ou padrão específico.

3.6.3. Arquitetura do agente

Seguindo as recomendações da FIPA, a JADE não procura limitar os tipos de agentes que podem ser implementados, ou seja, é totalmente neutro no que diz respeito à definição de um agente. A plataforma apenas reconhece um agente como um processo autônomo e independente que possui uma identidade e precisa se comunicar com outros agentes para atingir seus objetivos.

Em termos de implementação física, um agente da JADE opera como uma *thread* em que múltiplas tarefas são executas simultaneamente, dentre elas os comportamentos do agente e os procedimentos de comunicação. O agente é definido em uma classe Java chamada de `Agent` que serve de superclasse para qualquer outro agente construído. A classe `Agent` fornece os métodos e atributos necessários para a execução das tarefas básicas dos agentes, tais como:

- Transmissão e recepção de mensagens ACL por meio do objeto `ACLMessage`;
- Gerenciamento do ciclo de vida do agente, incluindo o estado de iniciar, suspender e desativar um agente;
- Escalonamento e execução de múltiplas tarefas concorrentes (comportamentos);
- Integração com as plataformas de agentes para a automação de tarefas rotineiras, como por exemplo, o registro no DF;
- Suporte à mobilidade e clonagem do agente em outros containeres utilizando a ontologia de mobilidade já disponível.

A Figura 3.8 denota a arquitetura interna de uma agente padrão na JADE. A parte superior mostra os comportamentos ativos que representam as ações e intenções do agente e são gerenciados pelo escalonador de comportamentos. É importante ressaltar que um único agente pode possuir diversos comportamentos, já que seu modelo computacional é multitarefa. Logo, cada funcionalidade ou serviço prestado por um agente deve ser modelado em termos de um ou mais comportamentos.

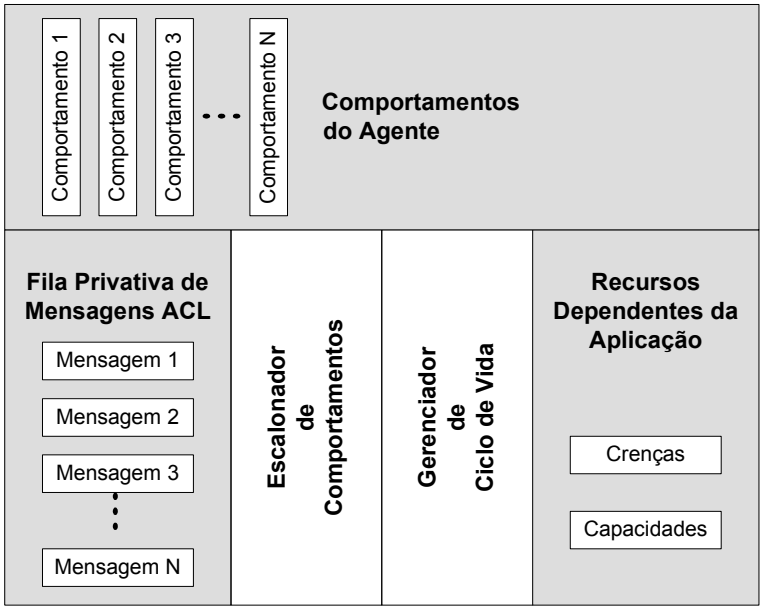


Figura 3.8 - Arquitetura de um agente JADE [55]

Na parte inferior, à esquerda da Figura 3.8, é mostrada a fila privativa de mensagens ACL. Independente do número de mensagens que chegam no agente, essas são empilhada e processadas uma a uma. Ao centro, tem-se o gerenciador de ciclo de vida que controla e alterna o estado atual do agente. À direita, são mostrados os recursos dependentes da aplicação como as crenças e capacidades do agente.

O agente em JADE pode ter seu estado alterado por meio do gerenciador de ciclo de vida. Os estados possíveis para o agente também seguem a proposta de ciclo de vida da FIPA, como foi mostrado anteriormente na Figura 3.5.

4 - CONCEPÇÃO DO *FRAMEWORK*

Os capítulos anteriores introduziram temas que serviram de apoio para o desenvolvimento da proposta principal desse trabalho. Recapitulando, foi visto que os SDIs (Sistemas de Detecção de Intrusão) são importantes componentes para a segurança de uma rede de computadores por fazerem a monitoração contínua do tráfego da rede e, também, dos processos executados nos servidores, revelando e bloqueando as ações dos atacantes. Esses sistemas são imprescindíveis, visto que o grande número de transações distribuídas, as constantes mudanças no ambiente e o grande volume de tráfego gerado impedem que o acompanhamento seja feito apenas pela intervenção humana.

Também que um SDI ideal precisa possuir alto desempenho para que consiga processar todos os dados que se encontram distribuídos pela rede e aplicar os algoritmos de detecção adequados. Dessa forma, poderá reagir rapidamente no caso de detecção de ataques. Além disso, é necessário que tenha um nível aceitável de acurácia para garantir baixos índices de falso-positivos e falso-negativos, aumentando, assim, a confiabilidade do sistema. Outro quesito diz respeito à capacidade de evolução do SDI que deve ser adaptável por meio da aprendizagem constante de seu ambiente e de novos padrões de ataques e, ainda, deve ser extensível para garantir a adição de novas funcionalidades e especializações.

Mais adiante, foi observado que o uso de agentes de software para apoio à segurança computacional traz diversos benefícios principalmente devido às características de autonomia, de mobilidade e da natureza distribuída pertinente a essa arquitetura. Esse modelo vem sendo bastante adotado, pois simplifica a operação por meio de um esquema baseado na delegação de tarefas e ainda permite a configuração e instalação remota de novos agentes no sistema, sem necessidade de paralisação.

Esse trabalho propõe uma estrutura para SDIs com uma arquitetura baseada na atuação de grupos de agentes especializados. Essa arquitetura possui uma configuração extremamente flexível que permite a implementação de diferentes estratégias de coleta de dados, análise, e resposta aos ataques por meio da extensão das classes básicas presentes no modelo. Por exemplo, para resolver um problema relativo ao processo de *port scanning,* o projetista estenderá o sistema a partir dos grupos de agentes básicos previstos pela arquitetura que farão todo o processo de coleta, descoberta de padrões e resposta aos ataques.

O presente capítulo tem o objetivo de apresentar esse *framework* que auxilia na construção de sistemas de detecção de intrusão fazendo uso de uma arquitetura orientada a agentes de software. Sendo um contexto tão diversificado, para que a proposta sirva de base para o desenvolvimento dos projetos de SDIs, deve oferecer os diversos serviços necessários, dando flexibilidade para o projetista e proporcionado vantagens para o próprio sistema em termos de desempenho, adaptação e extensão.

Como complementação desse capítulo, o Apêndice B fornece uma introdução às redes neurais artificiais para o entendimento de um dos componentes do *framework* e o Apêndice B sugere roteiros de instalação e configuração de todas as ferramentas tratadas e do próprio *framework*.

4.1 - CARACTERÍSTICAS DESEJÁVEIS

As características desejáveis para o *framework* podem ser agrupadas sob duas perspectivas: a do desenvolvimento e a do próprio produto – o sistema de detecção de intrusão. Na perspectiva do desenvolvimento, se almeja a melhoria do processo de construção, beneficiando os desenvolvedores, analistas e arquitetos dos projetos, por meio de metodologias, ferramentas e componentes de apoio. Já as características voltadas ao produto são as que refletem diretamente na atuação do SDI construído sobre o *framework* e podem ser classificadas entre características funcionais e não-funcionais como, por exemplo, a confiabilidade, a tolerância a falhas e a performance.

Como visto, o *framework* deve oferecer vantagens ao processo de construção dos SDIs cujos projetos são baseados em tecnologia de agentes de software. Nesse sentido, as características desejáveis para o *framework* são:

- Simplificar os mecanismos de processamento distribuído e concorrente necessários na detecção de intrusão;
- Acelerar os processos de concepção, construção, implantação e evolução do projeto;
- Propor um modelo mais natural para o desenvolvimento dos agentes, conduzindo a solução para que tenha alta capacidade de manutenção e reaproveitamento;
- Usar de forma transparente uma plataforma de agentes complacente com a FIPA;
- Suportar a aplicação de múltiplos algoritmos de detecção a fim de minimizar o índice de falsos positivos e falsos negativos
- Permitir tanto a implementação de técnicas baseadas no *host*, quanto baseadas na rede por meio de uma interface comum;
- Oferecer meios para que o processamento possa ser distribuído de forma arbitrária entre os servidores;
- Suportar o uso de ferramentas para a modelagem, criação e importação de ontologias para comunicação entre os agentes;
- Possibilitar uma comunicação consistente, até mesmo com agentes de outras plataformas, por meio do uso do padrão FIPA-ACL;
- Disponibilizar os componentes de serviços necessários para a validação do protótipo em dois cenários: detecção de usuários anômalos e detecção de *port scanning*.

Em relação ao produto construído, o *framework* visa oferecer uma arquitetura robusta que reflita em uma melhor atuação do sistema de detecção de intrusão. Crosbie e Spafford [30] definem algumas características essenciais para qualquer SDI, que se aplicam também ao *framework*:

- Deve ser tolerante a falhas e poder se recuperar em caso de panes acidentais ou causadas por atividades maliciosas

- Deve ser resistente à subversão, ou seja, deve ser capaz de avaliar seus próprios recursos para detectar e bloquear modificações maliciosas;
- Deve produzir baixa sobrecarga para minimizar o impacto nas operações normais dos sistemas monitorados;
- Deve ser flexível e configurável, permitindo, assim, uma adequação precisa das políticas de segurança e topologias das redes;
- Deve ter alta escalabilidade, possibilitando a boa operação em ambientes com um grande número de *hosts*;
- Deve possibilitar a degradação gradual do serviço em caso de falhas de alguns componentes do sistema, minimizando o impacto nos demais componentes;
- Deve possibilitar a reconfiguração dinâmica, permitindo que o administrador de segurança efetue atualizações na configuração sem a necessidade de reinicialização do sistema;

Adicionalmente, os seguintes requisitos são propostos para o produto baseado *framework*:

- Deve ser portável entre diferentes plataformas, permitindo, assim, a execução em ambientes heterogêneos;
- Deve ser um sistema híbrido para tirar proveito das vantagens de um SDI baseado no *host* (HIDS) e de um SDI baseado na rede (NIDS).

Ficando estabelecidos esses requisitos para o *framework* e para seus produtos, deve ser proposta uma arquitetura de software que atenda a essas necessidades. Como determinante, essa arquitetura deve ser baseada em agentes de software devido ao fato dessa abordagem já suprir intrinsecamente boa parte dessas necessidades.

4.2 - CONSIDERAÇÕES SOBRE A ARQUITETURA

A existência de diversos agentes trabalhando em conjunto em função do sistema sugere, inevitavelmente, um grau elevado de organização hierárquica e divisão do trabalho a fim de aumentar a eficiência dos processos. Dessa forma, um sistema de detecção desenvolvido com a estrutura proposta será decomposto em grupos de agentes, onde cada agente terá um papel bem definido. A cooperação entre os agentes de um determinado grupo possibilitará a resolução de um cenário de detecção em particular. O sistema completo de detecção será formado pelos diversos grupos existentes e seus respectivos agentes.

Esse contexto pode ser considerado uma metáfora em relação a algumas sociedades de insetos, como a das formigas. Esses interessantes seres se organizam em diferentes castas com a finalidade de delimitar as funções de cada indivíduo como mostrado na Figura 4.1. Cada casta é responsável por uma atividade específica dentro da colônia, como por exemplo, reprodução, alimentação ou segurança, e, uma vez definida a casta de um novo indivíduo, esse começa a operar incessantemente a fim de cumprir seu papel naquele grupo. A atuação conjunta de todas as castas garante a manutenção e desenvolvimento de todo ninho e, consequentemente, a proteção e perpetuação da espécie [54].

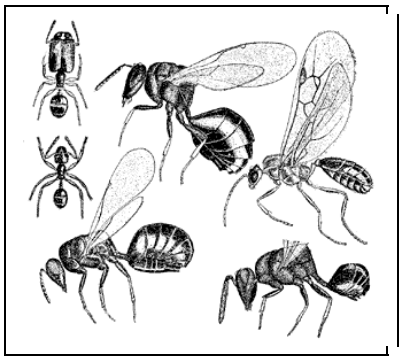


Figura 4.1 - Divisão em castas em uma colônia de formigas: soldados, zangões, operárias e rainha

O que tem chamado a atenção de alguns pesquisadores da computação distribuída é a forma como esses pequenos insetos conseguem resolver problemas complexos, que na maioria das vezes são incógnitos e imprevisíveis, de maneira extremamente natural tendo como base o cooperativismo. Cabe ainda ressaltar, que um único indivíduo não possui inteligência significativa, nem potencial para resolver todo problema, porém ao agirem em conjunto, o comportamento geral do grupo reflete certa inteligência.

A autonomia e mobilidade pertinentes também são bastante sugestivas, pois após a definição da função de cada indivíduo, esse passa a executá-la de forma autônoma, ou seja, não é necessária uma intervenção contínua a fim de garantir os resultados. Mesmo havendo imprevistos em seu percurso, procurará contorná-los, dentro de suas possibilidades, a fim de finalizar sua tarefa.

Não só as características de autonomia e a mobilidade dos agentes foram as responsáveis pela concepção do *framework*, pois esse tipo de proposta pode ser encontrado em diversos outros trabalhos. Na verdade, o próprio modelo organizacional dos componentes do *framework* foi o diferencial.

Ao invés de tratar esse problema de segurança em módulos ou camadas, como é de praxe, procurou-se organizar todo o esforço necessário em termos de funções bem definidas dentro de um grupo de agentes de software. Por exemplo, ao invés de fazer a proposta de um estágio de coleta do SDI aonde são exercidas as atividades de coleta de dados, foi proposto um modelo de agente de coleta que serviria para qualquer fim dessa natureza por meio de sua extensão.

Para se chegar nesse modelo foram estudadas algumas idéias do CIDF (*Common Intrusion Detection Framework*) [2] que serviram como um modelo de referência para uma especificação compatível com outros projetos. Houve também a colaboração do modelo genérico de SDI proposto no trabalho de Bace e Mell [3] e do trabalho de Bernardes [4] que apresenta um SDI com arquitetura baseada em camadas de agentes autônomos, além da própria noção de grupos (ou castas), com indivíduos especializados e móveis, abordada nessa seção.

4.3 - GRUPOS DE AGENTES

O modelo proposto sugere a definição de grupos formados por agentes especializados e propõe também uma diferenciação no nível cognitivo de cada tipo de agente. Sendo assim, alguns agentes terão maior capacidade de tomada de decisão enquanto outros apenas reagirão ao sistema com base em regras determinadas. A Figura 4.2 mostra a composição padrão de um grupo de agentes especializados e respectivas inter-relações que da arquitetura proposta.

Figura 4.2 –Modelo do grupo de agentes especializados

As funções possíveis para cada agente do grupo são descritas abaixo:

- **Agente de coleta:** responsável pela extração, transformação e carga dos dados brutos das fontes visitadas, colocando esses dados em um nível mínimo de entendimento para outros agentes;
- **Agente de análise:** após receber os dados do agente de coleta, fazem a detecção propriamente dita, por exemplo, com o auxílio de recursos de inteligência artificial ou se baseando em regras;
- **Agente de armazenamento:** armazena todas as informações necessárias de um cenário de detecção, como, por exemplo, os padrões de assinaturas dos ataques;
- **Agente de reação:** responsável pelo contra-ataque do sistema, podendo agir de forma passiva ou ativa;

- **Agente de coordenação:** cria e controla todos os agentes de um cenário de detecção e toma as decisões mais estratégicas do sistema, como a opção de reação em caso de detecção de intrusão.

Cada agente possui características específicas, dessa forma cada um trabalha em um escopo limitado e, ao necessitar de apoio em um aspecto que não pode tratar, solicita o auxílio ao agente responsável. Essas características específicas podem ser classificadas sobre o ponto de vista de autonomia, inteligência, visibilidade, capacidade de memória, mobilidade, complexidade de implementação, nível de atuação em relação à complexidade do conhecimento e cardinalidade no sistema, conforme a Tabela 4.1.

Tabela 4.1 – Comparativo entre os tipos de agentes

Característica	Agente de Coleta	Agente de Análise	Agente de Armazenamento	Agente De Reação	Agente De Coordenação
Autonomia	*Baixa*	*Média*	*Não possui*	*Baixa*	*Alta*
Inteligência	*Não possui*	*Alta*	*Não possui*	*Não possui*	*Alta*
Amplitude da Visão	*Estreita*	*Média*	*Não possui*	*Estreita*	*Alta*
Nível da Visão	*Operacional*	*Tática*	*Operacional*	*Operacional*	*Estratégica*
Capacidade de Memória	*Pequena*	*Pequena*	*Grande*	*Não possui*	*Média*
Duração da Memória	*Instantânea*	*Mais recente*	*Não volátil*	*Instantânea*	*Mais recente*
Mobilidade	*Alta*	*Configurável*	*Baixa*	*Alta*	*Baixa*
Complexidade de Implementação	*Baixa*	*Alta*	*Baixa*	*Baixa*	*Alta*
Nível de atuação	*Dados*	*Informações e Conhecimento*	*Dados*	*Parâmetros*	*Informações e Conhecimento*
Cardinalidade	*Muitos*	*Configurável*	*Poucos*	*Sob demanda*	*Poucos*

Como os agentes de coleta são meros extratores de dados brutos, possuem uma visão bastante limitada e operacional, ou seja, focada em seu objetivo específico, não possuindo inteligência nem autonomia fora do contexto de sua coleta, todavia podem concluir que certo pacote de coleta está corrompido e tentar refazê-lo. Também têm pouca capacidade de armazenamento e possuem memória volátil, já que repassam as coletas para agentes superiores e devem ser "leves" o suficiente para que o grande número desses agentes não impacte no desempenho da rede. As coletas, bem como toda informação trocada pelos agentes, são especificadas por meio de ontologias apropriadas para cada tipo de mensagem, tornando essa informação nativa para o sistema.

Os agentes de análise já apresentam bastante inteligência, principalmente para a tomada de decisão em relação às coletas recebidas, porém sua autonomia fica limitada aos objetivos gerais do sistema demandados pelos agentes de coordenação, por isso possuem uma visão um pouco mais ampla, porém em nível tático. Possuem uma maior capacidade de memória e persistência, já que muitas vezes precisam analisar o histórico de ocorrências para concluir sua análise. Podem se apresentar em menor número e não necessitam de tanta mobilidade, podendo ser estacionários principalmente devido a capacidade de processamento de múltiplas coletas.

Os agentes de armazenamento agem basicamente como repositórios de dados para a leitura e gravação de informações, por isso seu enfoque é no alto poder de armazenamento e persistência. Sua mobilidade também pode ser mínima e usada apenas para a replicação das bases em outros pontos da rede, fazendo uma espécie de contingência.

Os agentes de reação são acionados apenas após a detecção de ataques na rede e devem fazer a proteção e/ou divulgação do problema. Assim como os coletores, os agentes de reação simplesmente seguem ordens, agindo em um nível mais operacional. Em relação ao número no sistema, é bastante relativo, visto que ataques coordenados podem exigir um maior número de agentes distribuídos na rede.

Os agentes de coordenação podem ser considerados como os mais inteligentes em termos estratégicos e, por isso, possuem maior visão do ambiente, percebendo os ataques nos diversos pontos da rede e coordenando mecanismos de defesa envolvendo um maior número de agentes. Assim como os agentes de análise, necessitam de maior poder de armazenamento e persistência para que os algoritmos de reconhecimento de padrões sejam atendidos. O número de agentes de coordenação será proporcional ao número de grupos de agentes e a sua replicação pode ser útil para aspectos de contingência e aumento do desempenho, pois possui a habilidade de clonar agentes que foram desativados por falhas no sistema.

Um SDI construído a partir do *framework* será formado pelos diversos grupos desenvolvidos. Cada grupo é na verdade uma extensão do grupo básico presente no *framework* e deve tratar um cenário específico de segurança. A existência dos diversos grupos atuando conjuntamente caracterizará o próprio sistema de detecção de intrusão. A Figura 4.3 ilustra um SDI capaz de tratar usuários anômalos, *port scanning* e DDoS por meio da implementação de um grupo de agentes especializados para cada cenário.

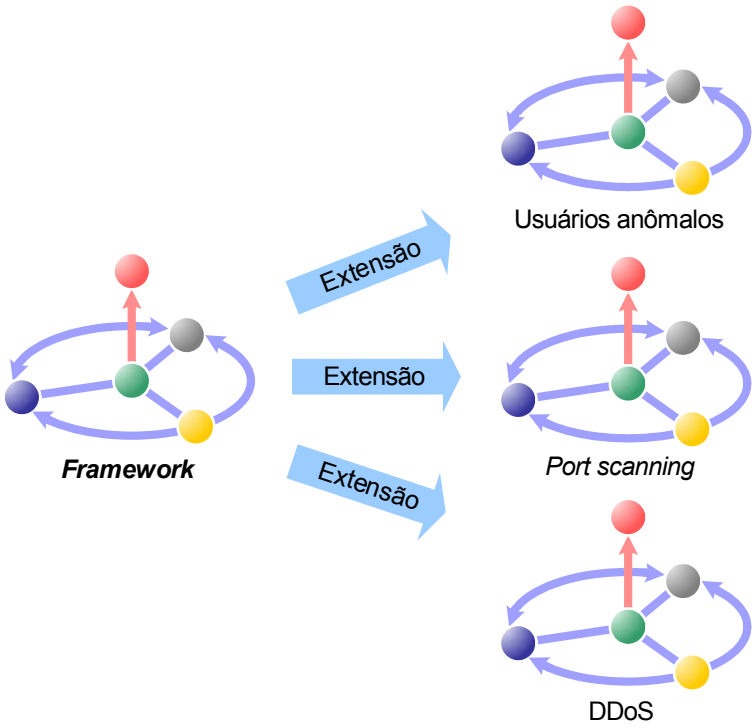


Figura 4.3 - SDI formado pelos grupos de agentes especializados

4.4 - PLATAFORMA DE AGENTES

Como foi visto, o *framework* promove a criação de uma sociedade de agentes especializados que cooperam entre si para a resolução dos problemas, formando assim o SDI. Para isso, os agentes devem habitar em um ambiente de execução de agentes que já se responsabiliza por alguns serviços gerais, tais como o controle transacional, a mobilidade e a comunicação entre agentes.

A plataforma de agentes escolhida para o sistema foi a JADE (*Java Agent Development Environment*) que se baseia completamente em tecnologia Java e vem sendo bastante utilizada para aplicações acadêmicas e comerciais envolvendo agentes de software.

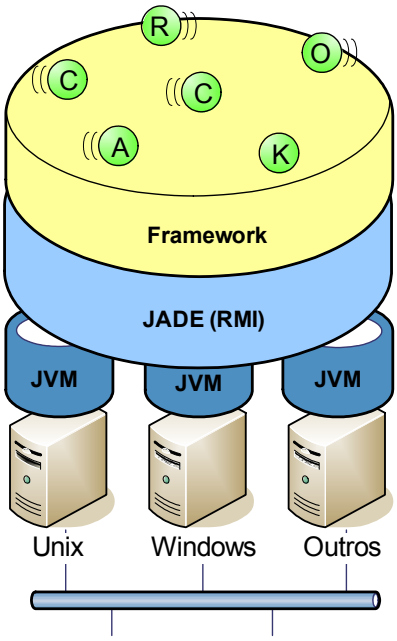


Figura 4.4 – Plataforma de agentes do *framework*

Conforme a Figura 4.4, cada *servidor* deve ter uma JVM (*Java Virtual Machine*) em execução para que os containeres de agentes do JADE, que são os microambientes de execução, possam ser criados. Usando o protocolo RMI (*Remote Method Invocation*) para configurar o canal de comunicação, os diversos containeres podem formar um só ambiente de execução ou macroambiente que interliga todos os *hosts*. Sendo assim, cada agente tem ao seu alcance todos os outros containeres e respectivos agentes presentes no macroambiente de forma direta e transparente, sem precisar conhecer a estrutura de rede de comunicação existente. Por exemplo, caso o macroambiente esteja criado, para que um agente se movimente de um ponto ao outro da rede não é preciso conhecer o endereço IP do destino e rota para alcançá-lo.

Sobre a plataforma de agentes atuam os grupos de agentes especializados do *framework,* formando a camada mais superficial. Nesse âmbito, os agentes têm liberdade de movimentação e comunicação entre os *containers*. Cada *servidor* deve iniciar seu próprio *container* a fim de permitir a atuação do sistema de detecção.

4.5 - ESTRATÉGIAS DE ATUAÇÃO

Uma vez que os agentes implementados pelo *framework* são portáveis e móveis, podem atuar de diversas maneiras de acordo com as definições atribuídas pelo agente de coordenação. Por exemplo, pode ser definido que apenas o agente de coleta visitará cada servidor e enviará os dados coletados para um container central aonde residem os outros agentes. Da mesma maneira, pode ser definido que todos os agentes residirão em um único container e farão todo o processamento com base no tráfego da rede. Essa flexibilidade proposta pelo *framework* o torna bastante adaptável às características de cada ambiente.

Algumas estratégias de atuação possíveis são mostradas na Figura 4.5. Em (a) é mostrado um modelo baseado no *host* (HIDS) com um servidor central que faz todo processamento e controle dos agentes de coleta espalhados pelos *hosts*. Em (b) é mostrado um modelo baseado na rede (NIDS) onde todo tráfego da rede é capturado em um servidor central aonde residem todos os agentes. Em (c) é mostrado um modelo totalmente distribuído e baseado no *host*, podendo haver trocas de informações entre os grupos de diferentes *hosts* para fins de atualização. Por fim, em (d), é mostrada uma distribuição análoga ao padrão de sistemas em três camadas (*three-tiers*) – servidor de banco de dados, servidor de processamento e *host* cliente – que propõe um alto desempenho para cada estágio do *framework*.

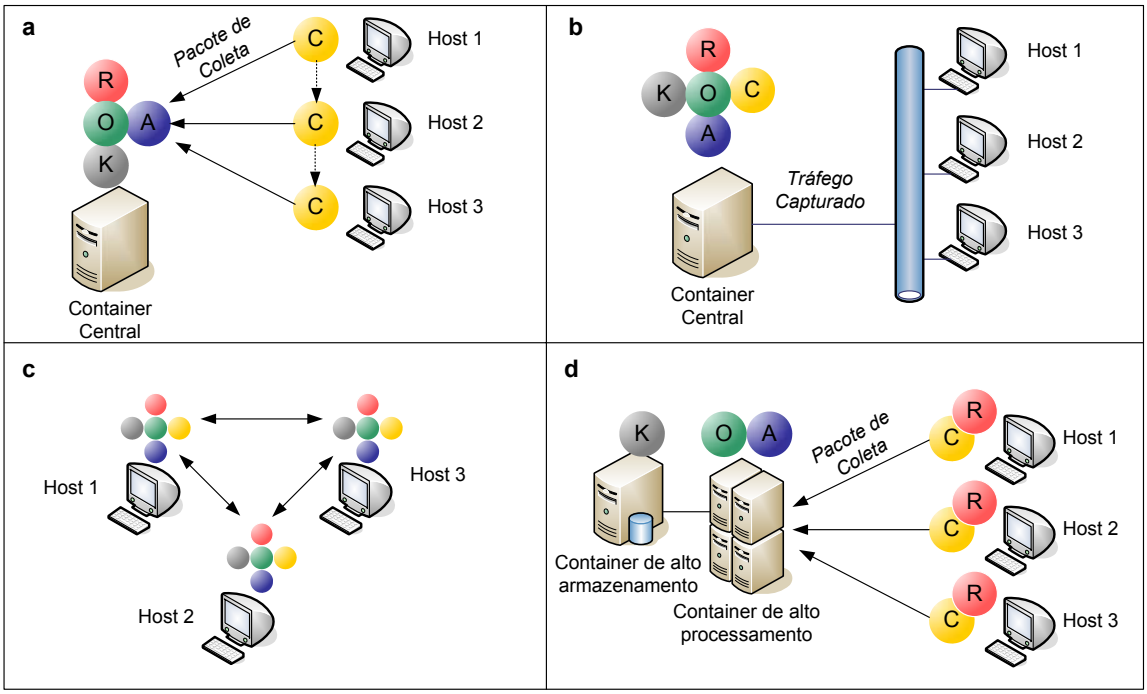


Figura 4.5 - Estratégias de atuação do SDI sobre o *framework*

4.6 - MODELOS DOS AGENTES

Os agentes especializados foram modelados a partir da classe de agente padrão (`Agent`) que contém os serviços básicos da própria plataforma de agentes (JADE). Um agente do JADE possui métodos e atributos de controle e, ainda, permite a inclusão dos comportamentos desejados para sua execução. Conforme esquematizado pela Figura 4.6, as demais classes são derivadas da *Agent* e foram implementadas para agregar os serviços necessários ao SDI.

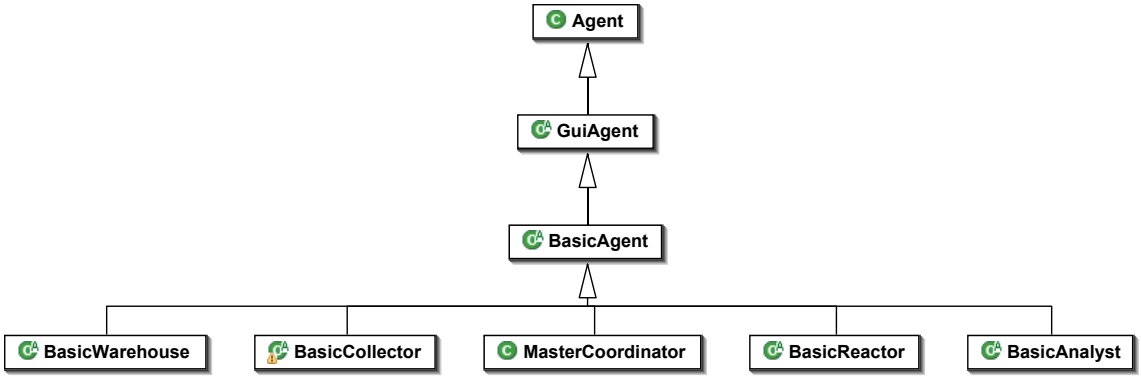


Figura 4.6 - Hierarquia das classes do *framework*

A classe `BasicAgent` representa a entidade de agente mais genérica que serve de base para os outros agentes do *framework*. Ela implementa os serviços comuns de todos agentes, como por exemplo, a ativação da interface gráfica e o registro automático do agente no *Directory Facilitator*. A Figura 4.7 mostra, à esquerda, o modelo da classe do agente básico e, à direita, seus respectivos serviços e comportamentos.

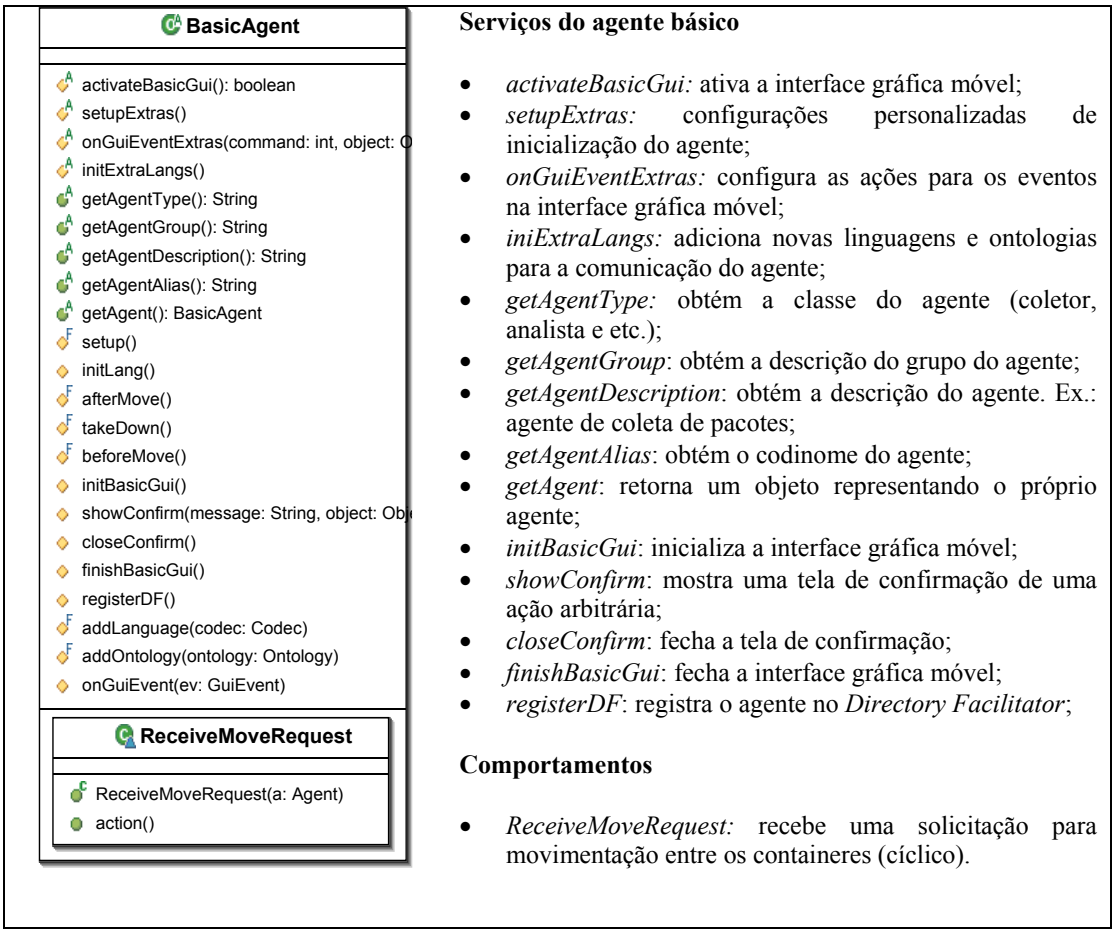


Figura 4.7 - Classe do agente básico - BasicAgent

Nas próximas seções serão abordados os modelos de cada agente especializado que derivam da classe `BasicAgent`. Serão vistas três visões de cada modelo – modelo de caso de uso estendido para agentes, modelo de sequência que representa cada comportamento do agente e, por fim, o modelo da estrutura estática da classe. A diagramação dos modelos foi possível graças ao uso da AUML (*Agent-based Unified Modeling Language*) que é uma extensão da UML convencional padronizada pela FIPA para atender as necessidades dos projetos envolvendo agentes de software [46].

4.6.1. Modelo do agente de coordenação

Conforme mostrado na Figura 4.8, os casos de uso do agente de coordenação são: “Solicitar Reação”, “Solicitar Coleta” e “Receber Alarme”. Como pode ser visto, essa visão do modelo utiliza as notações do diagrama de casos de uso da UML, só que utilizando um novo estereótipo `<<agent>>` que permite a extensão desse diagrama para representar agentes de software. Esse diagrama procura descrever objetivamente as funcionalidades do agente, sem se preocupar com a implementação interna de cada função.

O caso de uso “Solicitar Coleta” é responsável por acionar o agente de coleta daquele grupo para iniciar a captura de informações de um *host* especificado. O caso de uso “Receber Alarme” obtém pela os alarmes do agente de análise. O caso de uso “Solicitar Reação” aciona o agente de reação para que intervenha em um determinado *host*, geralmente, após o recebimento do alarme.

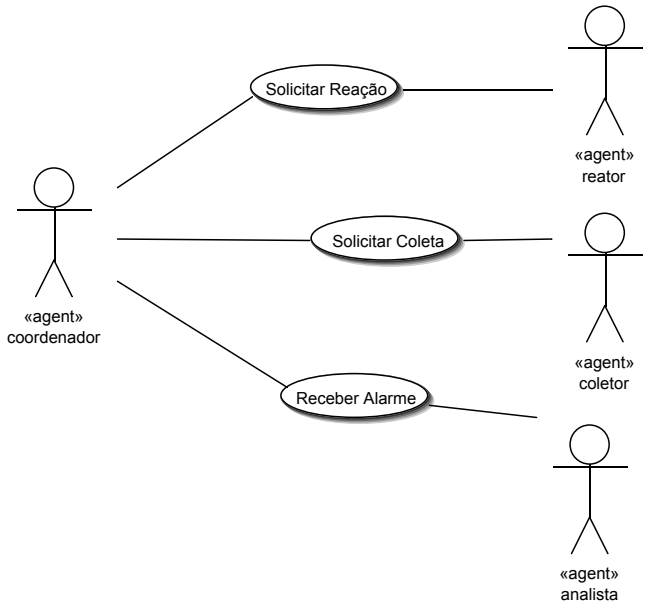


Figura 4.8 - Casos de uso do agente de coordenação

Os comportamentos previstos para o agente de coordenação podem ser modelados por meio do fluxograma mostrado na Figura 4.9. Essa visão do modelo usa os diagramas de atividades da UML que denotam a seqüência dos processos que compõe o comportamento.

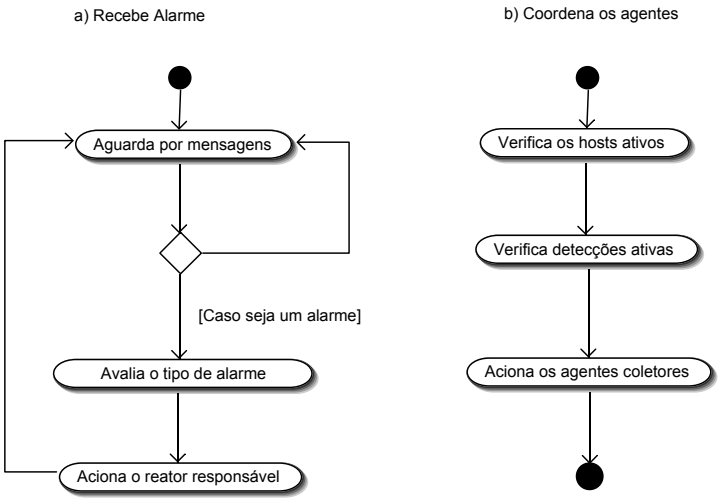


Figura 4.9 - Comportamentos do agente de coordenação

O comportamento "Recebe Alarme" é responsável pelo recebimento de mensagens de alarme do agente de análise e se caracteriza por um processo cíclico que avalia a cada instante as novas mensagens, sendo possível também o empilhamento no caso da ocorrência de inúmeros alarmes. Existe ainda um filtro que permite avaliar se o alarme enviado pertence àquele cenário coordenado pelo agente, evitando um processamento inútil. Esse procedimento é feito com avaliação da ontologia de origem do alarme, por exemplo, um alarme que pertença a uma ontologia para detecção de usuários anômalos não poderá ser tratado por um agente que opera com uma ontologia para detecção de *port scanning*. Após a filtragem, o agente de coordenação executa uma contramedida que é feita em parceria com um agente de reação.

O comportamento "Coordena os agentes" se resume em três processos básicos: verificação dos *hosts* que serão monitorados pelos agentes, verificação dos mecanismos de detecção que estarão habilitados (detecção de usuários anômalos, detecção de *port scanning*, etc.) e iniciação da detecção por meio do encaminhamento dos agentes coletores.

A Figura 4.10 mostra a classe modelada para o agente de coordenação. Como pode ser visto, esse agente não possui métodos específicos além dos definidos pelo agente básico. Em relação aos comportamentos, são tratados pelas classes internas (*inner classes*) `Request Reaction` e `Receive Alarm`. O método `action()` de ambas executa o comportamento após a criação do agente.

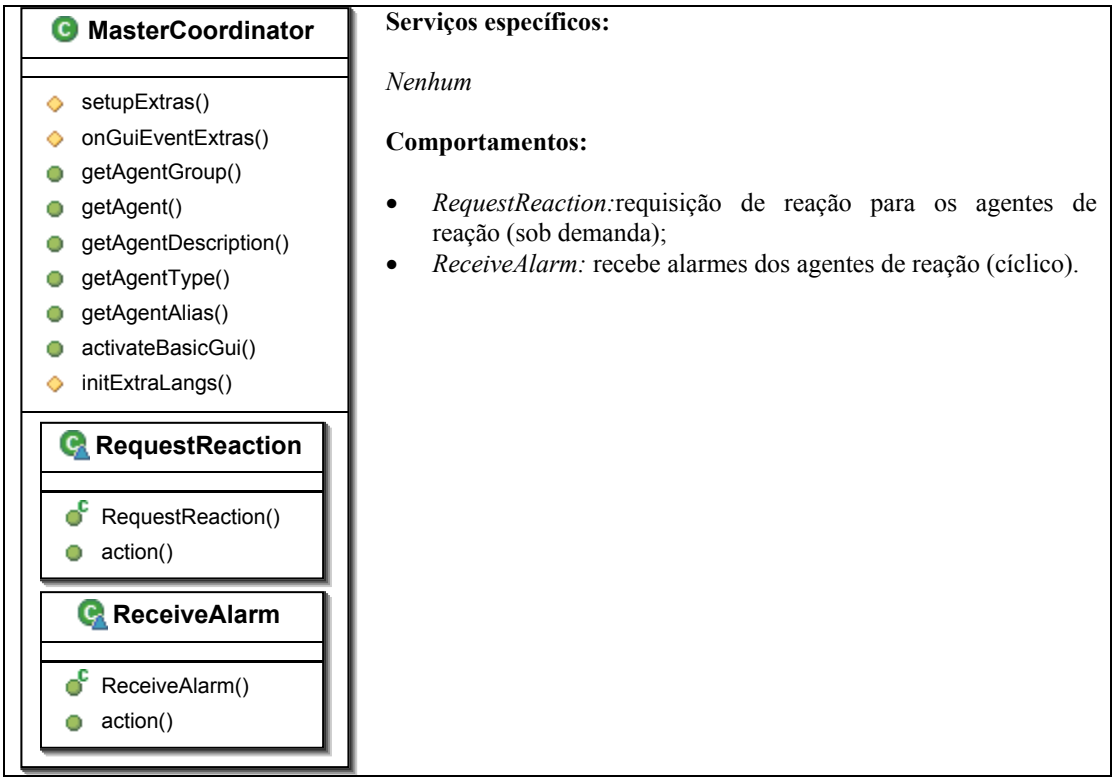


Figura 4.10 - Classe do agente de coordenação - MasterCoordinator

É importante ressaltar que para essa primeira versão do *framework* não estão inclusos os mecanismos de inteligência artificial que permitam ao agente de coordenação tomar decisões com base em histórico e no próprio contexto do ambiente, se baseando puramente em regras definidas pelo administrador. Ou seja, a atuação do agente de coordenação não é autônoma, mas, sim, baseada em roteiros e regras. Dessa forma, esse agente é idêntico em qualquer grupo de agentes especializados, podendo ser compartilhado entre os diferentes cenários de detecção de intrusão.

4.6.2. Modelo do agente de coleta

A Figura 4.11, mostra os casos de uso do agente de coleta. O caso de uso "Receber Requisição de Coleta" trata as solicitações do agente de coordenação e inclui o caso de uso "Mover para *Host*", caso o agente não esteja ainda no *host* alvo da coleta. O caso de uso "Processar Coleta" executa o processo de coleta propriamente dito. O caso de uso "Enviar Coleta" transmite os dados coletados para o agente de análise daquele grupo de agentes.

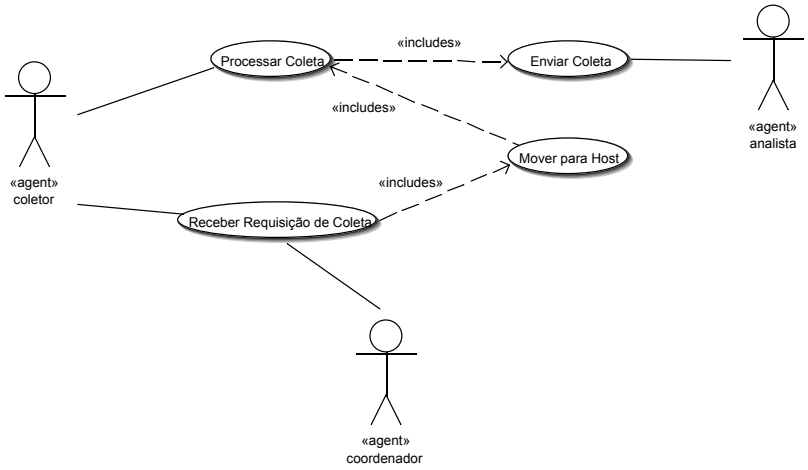


Figura 4.11 - Casos de uso do agente de coleta

Esse agente possui dois comportamentos: o comportamento "Recebe Requisição de Coleta" e o comportamento "Processa e Envia Coleta", como mostrado na Figura 4.12. O comportamento "Recebe Requisição de Coleta" (a) é cíclico e faz a filtragem das mensagens adequadas, após isso move o agente para o local especificado pelo agente de coordenação. O comportamento "Processa e Envia Coleta" (b) executa a coleta e envia os dados tratados para o agente de análise.

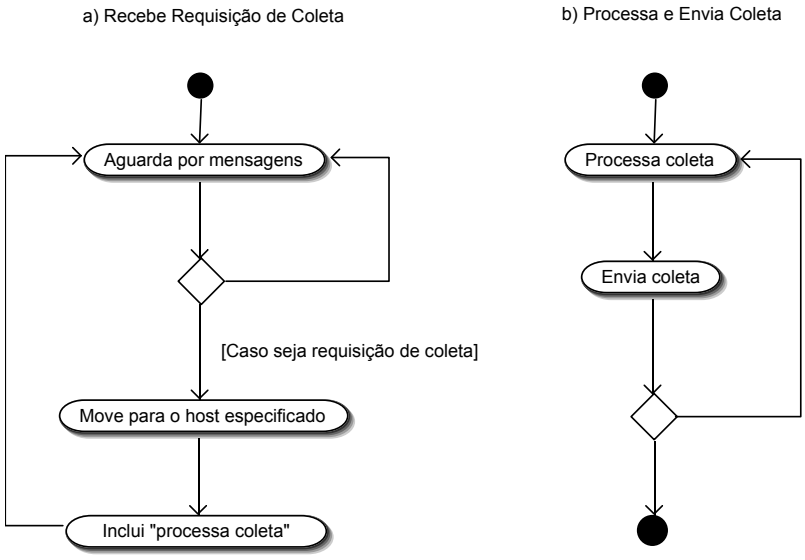


Figura 4.12 - Comportamentos do agente de coleta

A Figura 4.13 mostra a implementação física da classe do agente de coleta e seus respectivos serviços e comportamentos. O método específico `collect()` é o responsável pela definição do procedimento de coleta, por exemplo, leitura de um arquivo de *log*, captura de pacotes da rede ou, mesmo, monitoramento contínuo das ações dos usuários. Os comportamentos são descritos em termos das classes internas `ReceiveCollectRequest`, `ProcessCollect` e `SendCollect`.

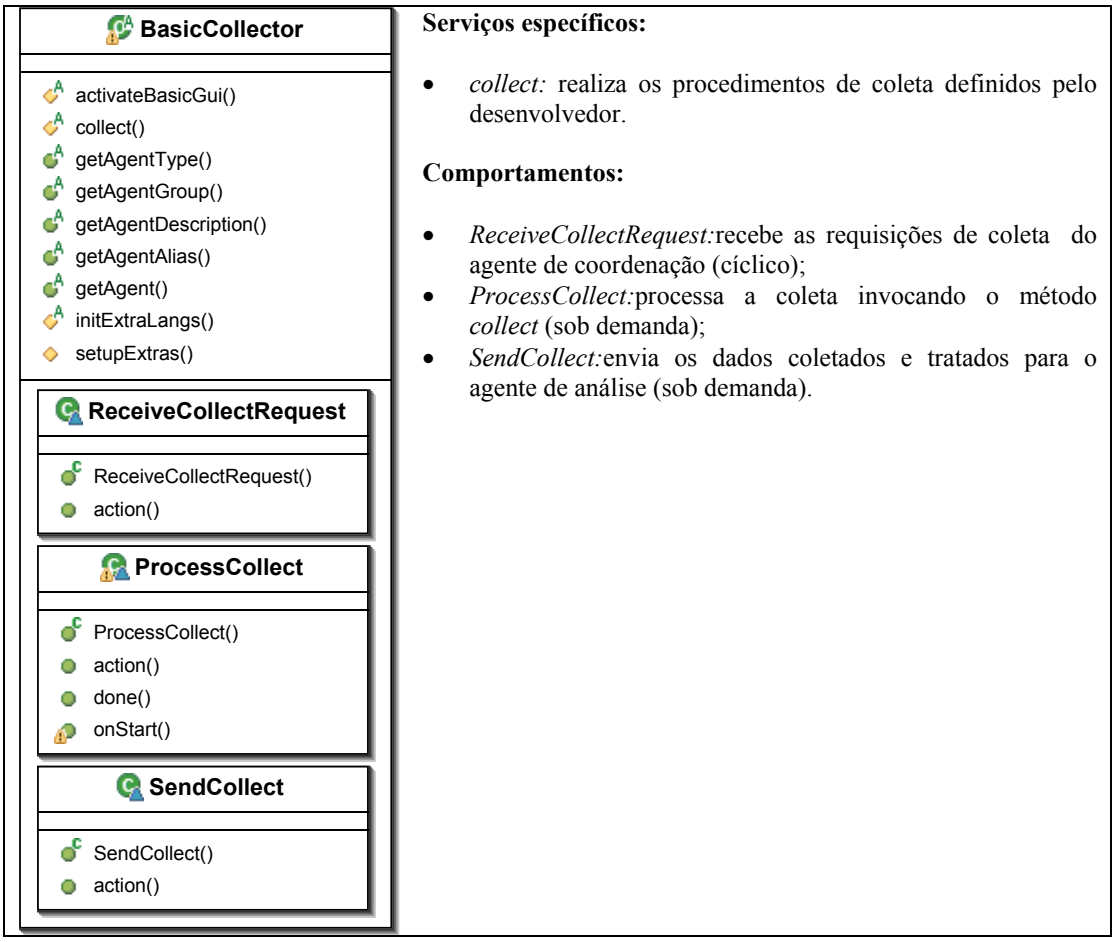


Figura 4.13 - Classe do agente de coleta

4.6.3. Modelo do agente de análise

Como pode ser visto na Figura 4.14, o agente de análise com certeza é o componente mais complexo do grupo de agentes, isso até que sejam implementadas as questões de autonomia do agente de coordenação. Para o bom funcionamento do agente de análise precisa estar sempre atualizado com os últimos padrões e conhecimentos adquiridos pelo sistema como um todo, por isso da existência do caso de uso "Solicitar Conhecimento". Além disso, precisa implementar algoritmos confiáveis para discernir entre uma ação anômala e normal por meio do caso de uso "Analisar Coleta" e, ainda, enviar alarmes para o agente de coordenação do grupo, quando for confirmada uma intrusão, por meio do caso de uso "Enviar Coleta".

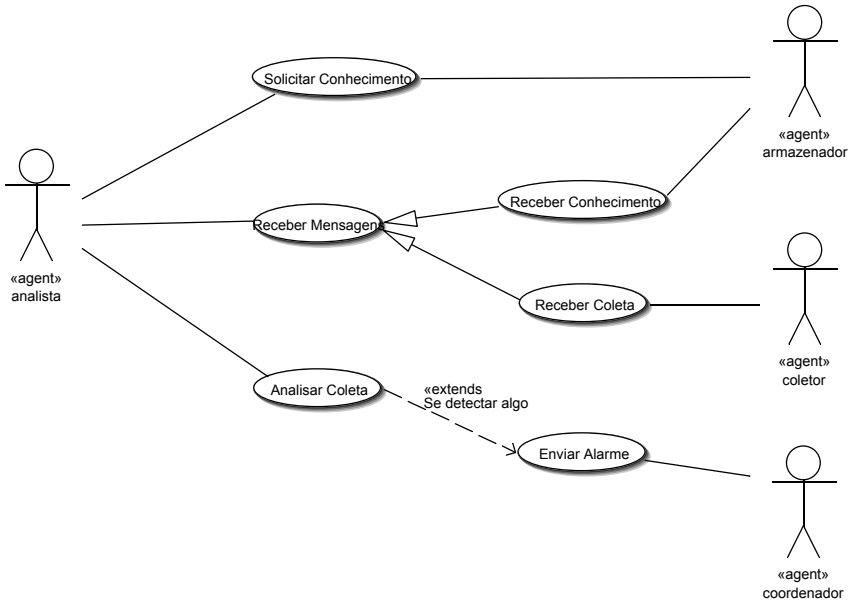


Figura 4.14 - Casos de uso do agente de análise

Os comportamentos do agente de análise podem ser observados na Figura 4.15. O comportamento "Recebe Coleta" verifica continuamente novas mensagens filtrando aquelas de interesse e compatíveis com a ontologia do grupo. Caso a nova mensagem seja uma atualização de conhecimento (a), o comportamento "Atualiza Conhecimento" é iniciado, caso a mensagem seja uma coleta, é iniciado o comportamento "Analisa Coleta" (a). Finalmente, se alguma anomalia for detectada é iniciado o comportamento "Envia Alarme" (b).

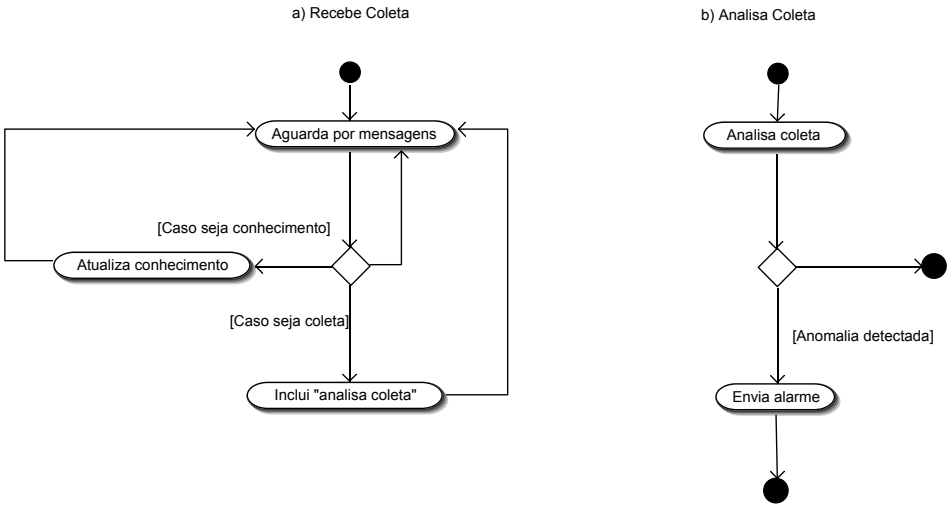


Figura 4.15 - Comportamentos do agente de análise

A implementação física da classe do agente de análise é mostrada na Figura 4.16. O método `analyze()` é responsável pela definição do processo de análise, devendo ser aplicada nesse ponto toda a inteligência necessária (redes neurais, redes bayesianas, classificadores e etc.) para que o agente tome as decisões corretamente. O método `getMyWarehousingAgentName`, `getMyKnowledgeRequest`, `isMyCollect` e `isMyKnowledge` são apenas filtros para otimizar todo processo. Os comportamentos já descritos são definidos em termos das classes internas `ReceiveMessages`, `AnalyseCollect`, `UpdateKnowledge` e `SendAlarm`.

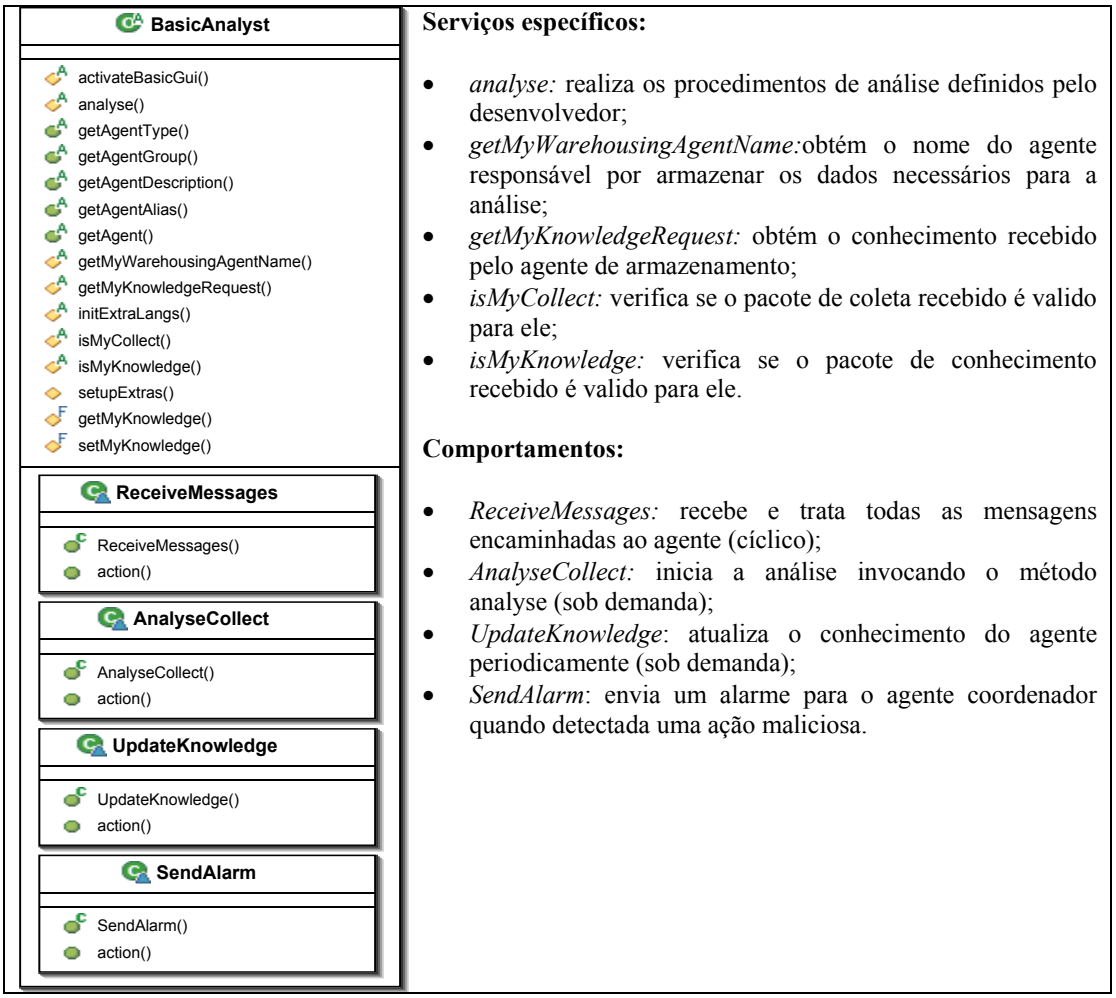


Figura 4.16 - Classe do agente de análise

4.6.4. Modelo do agente de reação

A Figura 4.11, mostra os casos de uso do agente de coleta. O caso de uso "Receber Requisição de Reação" trata as solicitações do agente de coordenação e inclui o caso de uso "Mover para *Host*", caso o agente não esteja ainda no *host* alvo do ataque. O caso de uso "Processar Reação" executa o processo de reação propriamente dito e, geralmente, terá que interagir com outros sistemas e serviços do *host* de forma nativa a fim de bloquear a ação do atacante. Um exemplo disso seria o bloqueio daquela conexão no *firewall* ou desativação da conta de usuário inválido. Como o desenvolvedor deverá sobrescrever esse método poderá direcionar a solução para um modelo ativo ou passivo.

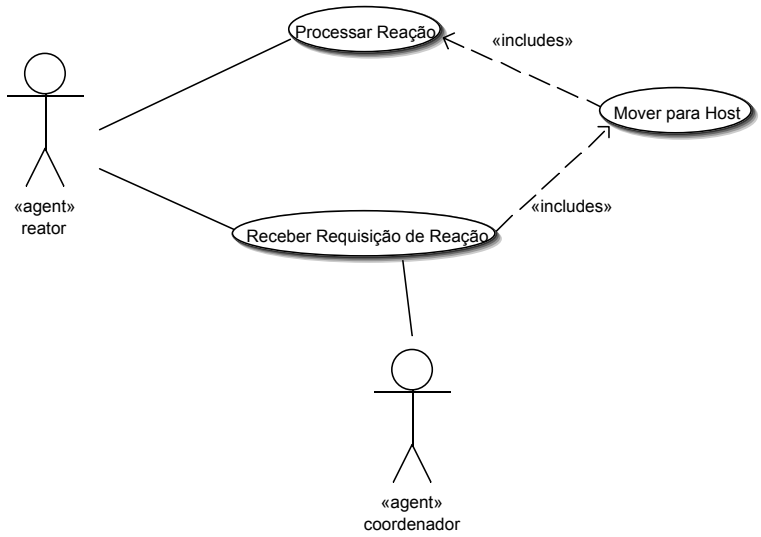


Figura 4.17 - Casos de uso do agente de reação

Como mostrado na Figura 4.18, o agente de coleta possui dois comportamentos: o comportamento "Recebe Requisição de Coleta" e o comportamento "Processa e Envia Coleta". O comportamento "Recebe Requisição de Reação" (a) é cíclico e faz a filtragem das mensagens adequadas, após isso move o agente para o local especificado pelo agente de coordenação. O comportamento "Processa Reação" (b) realiza o processo de reação especificado.

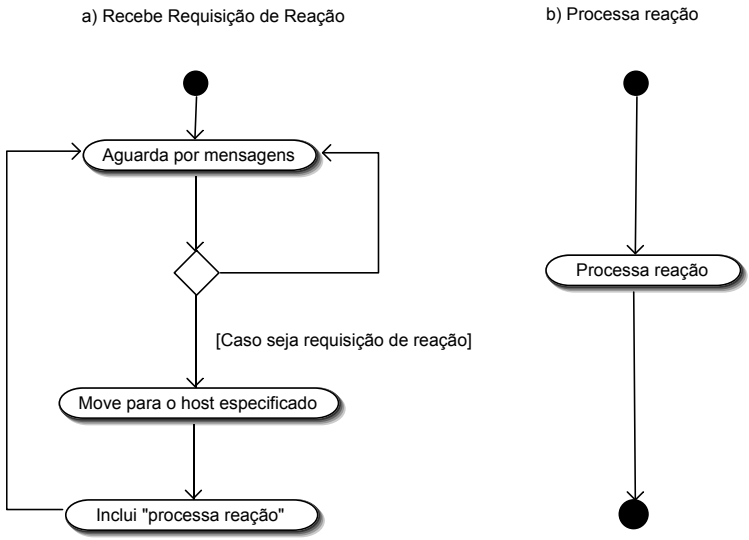


Figura 4.18 - Comportamentos do agente de reação

A Figura 4.19 mostra a implementação física da classe do agente de reação e seus respectivos serviços e comportamentos. O método específico `react()` é o responsável pela definição do procedimento de reação, por exemplo, ativação de uma regra no *firewall*, bloqueio da conta do usuário inválido ou, mesmo,desativação de um determinado *host*. Os comportamentos são descritos em termos das classes internas `ReceiveReactionRequest` e `ProcessReaction`.

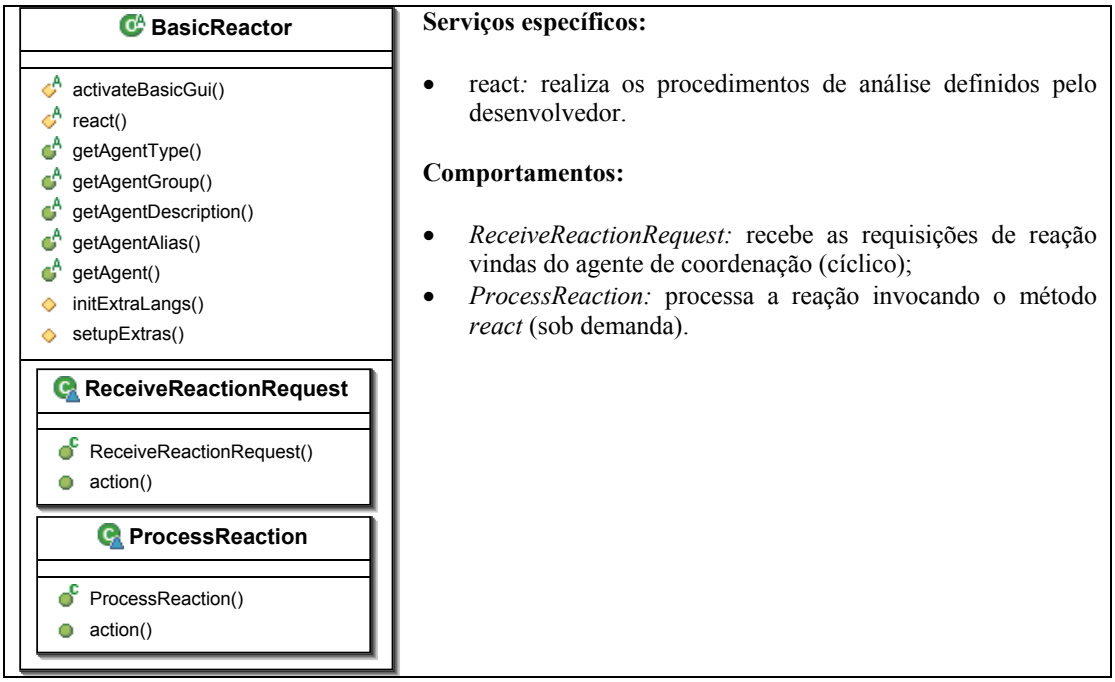


Figura 4.19 - Classe do agente de reação

4.6.5. Modelo do agente de armazenamento

A Figura 4.20, mostra os casos de uso do agente de armazenamento. O caso de uso "Receber Requisição de Conhecimento" trata as solicitações do agente de análise. O caso de uso "Consultar Conhecimento" executa o processo de obtenção do conhecimento, podendo interagir com bases de dados em diversos formatos e aplicar algoritmos de *data mining* para extrair conhecimentos indiretos. O caso de uso "Enviar Conhecimento" faz o envio do conhecimento adquirido para o agente de análise seguindo a taxonomia daquele grupo de agentes.

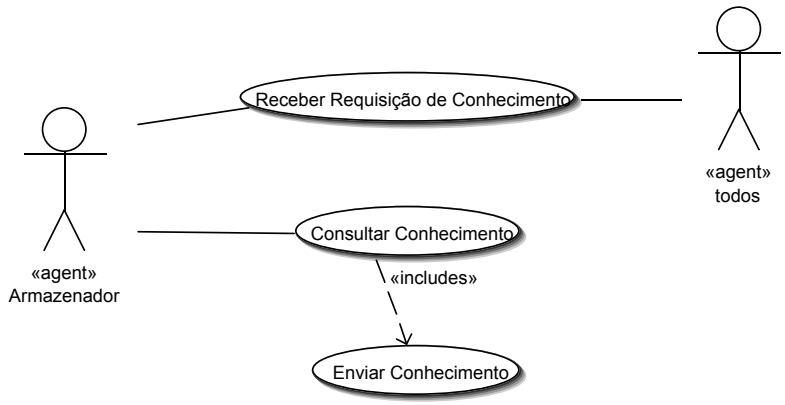


Figura 4.20 - Casos de uso do agente de armazenamento

Como mostrado na Figura 4.21, o agente de coleta possui dois comportamentos: o comportamento "Recebe Requisição de Conhecimento" e o comportamento "Envia Conhecimento". O comportamento "Recebe Requisição de Conhecimento" (a) é cíclico e faz a filtragem das mensagens adequadas e executa o processo de obtenção daquela informação. O comportamento "Envia Conhecimento" (b) envia por meio de mensagens o conhecimento para o agente de análise.

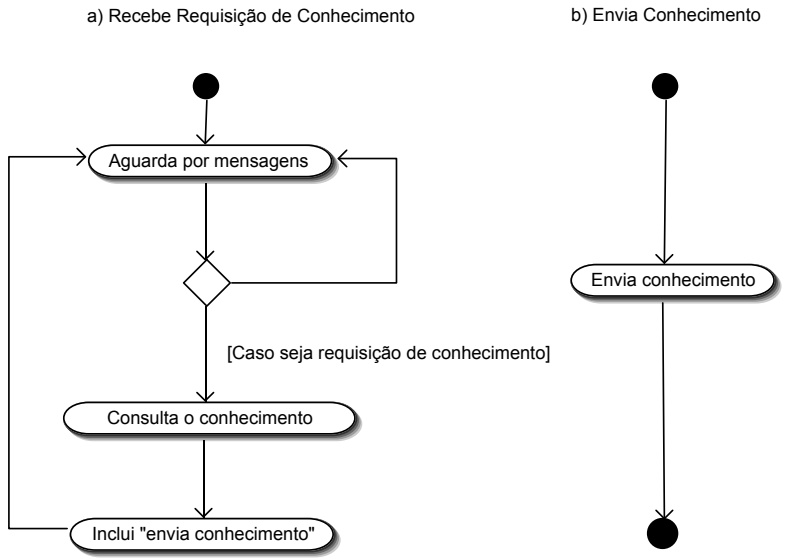


Figura 4.21 - Comportamentos do agente de armazenamento

A Figura 4.22 mostra a implementação física da classe do agente de armazenamento e seus respectivos serviços e comportamentos. O método específico `query()` é o responsável pela definição do procedimento de obtenção da informação, por exemplo, consulta a uma base de usuários válidos, *checksums* de arquivos do sistema ou a configuração de uma rede neural treinada para reconhecer *port scanning*. Os comportamentos são descritos em termos das classes internas `ReceiveKnowledgeRequest`, `QueryKnowledge` e `SendKnowledge`.

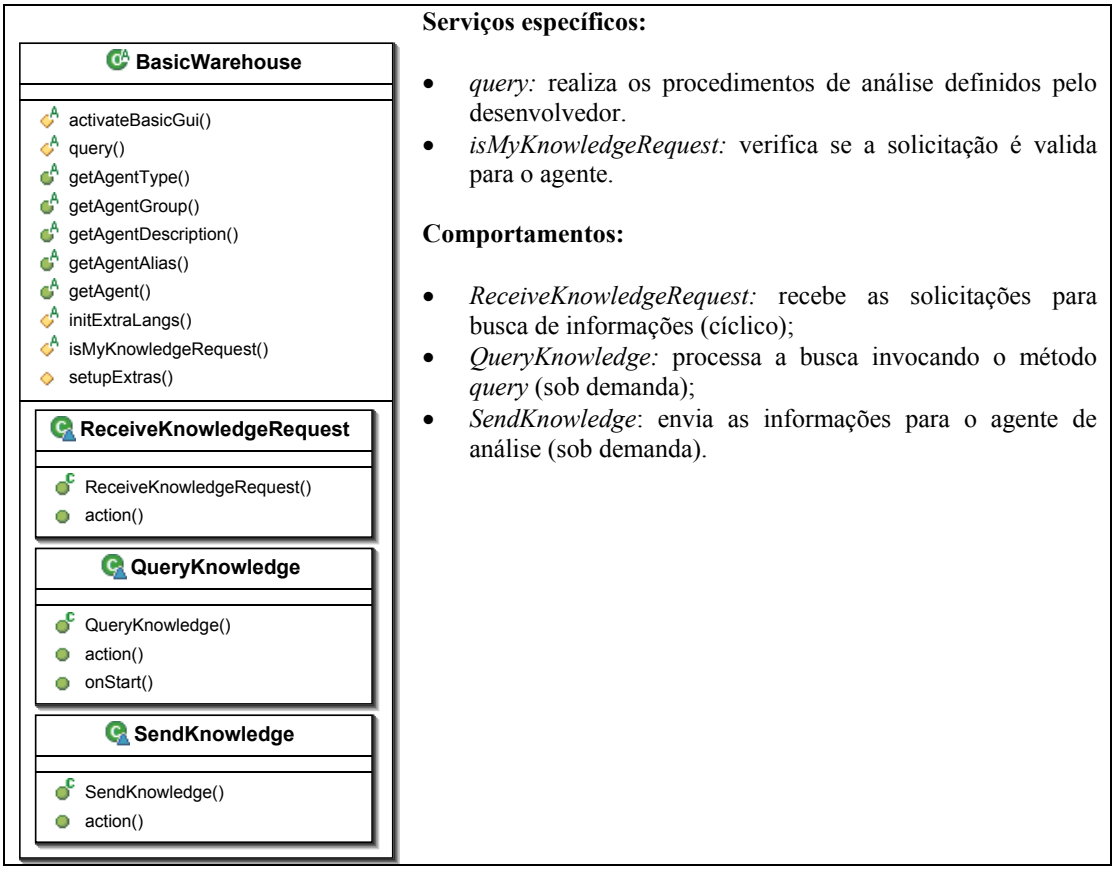


Figura 4.22 - Classe do agente de armazenamento

4.7 - ORGANIZAÇÃO DO FRAMEWORK

A intenção dessa seção é mostrar ao desenvolvedor como o *framework* é organizado fisicamente para que as novas implementações possam seguir essas diretrizes. Ao ser criado um novo grupo de agentes, deverão ser adicionados novos arquivos e bibliotecas seguindo essa orientação.

As próprias convenções de código Java recomendam que um sistema seja organizado em termos de pacotes, por isso o *framework* define uma série desses para agrupar as diversas funções do sistema de acordo com sua aplicação. A Tabela 4.2 mostra a organização desses pacotes e descreve o tipo de conteúdo presente nos mesmos.

Tabela 4.2 - Organização dos pacotes do *framework*

Endereço do pacote	Conteúdo
`br.unb.ids`*`framework`*	Recursos globais
`br.unb.ids`*`framework`*`.agents`	Definição dos tipos básicos de agentes
`Br.unb.ids`*`framework`*`.groups`	Definição de cada grupo de agentes especializados
`br.unb.ids`*`framework`*`.ontologies`	Definição das ontologias
`br.unb.ids`*`framework`*`.share`	Bibliotecas de apoio e utilitários
`br.unb.ids`*`framework`*`.ui`	Interfaces gráficas (GUI)
`br.unb.ids`*`framework`*`.agents.analysts`	Classes dos agentes de análise
`br.unb.ids`*`framework`*`.agents.collectors`	Classes dos agentes de coleta
`br.unb.ids`*`framework`*`.agents.coordinators`	Classes dos agentes de coordenação
`br.unb.ids`*`framework`*`.agents.reactors`	Classes dos agentes de reação
`br.unb.ids`*`framework`*`.agents.warehouses`	Classes dos agentes de armazenamento
`starlight.util`	Utilitários de codificação e decodificação em Base64
`jade.mtp.http`	Protocolos de interoperação entre containers baseado em HTTP
`jade.mtp.iiop`	Protocolos de interoperação entre containers baseado em IIOP
`jade`	Classes da plataforma JADE
`javax.activation`	Classes da biblioteca de ativação de *javabeans*
`javax.mail`	Classes da biblioteca de geração e recepção de e-mail
`org.joone`	Classes da biblioteca de redes neurais artificiais
`net.sourceforge.jpcap`	Classes da biblioteca de captura de pacotes
`junit`	Classes da biblioteca de testes unitários

Alguns pacotes serão amplamente mantidos pelo desenvolvedor, como o `br.unb.ids`*`framework`*`.ontologies` que define as ontologias e taxonomias de cada cenário de detecção de intrusão. Outro importante pacote é o `br.unb.ids`*`framework`*`.agents.*` que define os próprios grupos de agentes especializados. A fim de oferecer alguns serviços adicionais para a implementação de cenários de prova de conceito, ainda foram adicionados pacotes externos como o `javax.mail`, `org.joone` e `net.sourceforge.jpcap`.

A Figura 4.23 mostra a configuração física do *framework*. O *framework* é empacotado em um *Java Archive* (`ids`*`framework`*`.jar`) que acessa os arquivos de configuração (`Parameters.properties`), idiomas (`Message_*.properties`) e as bibliotecas externas necessárias. Em tempo de execução, o *framework* faz uso do próprio JADE (`jade.jar`) e das bibliotecas de *runtime* do Java (`rt.jar`). Cada grupo de agentes poderá ser empacotado separadamente como no `ids_exemplo.jar`.

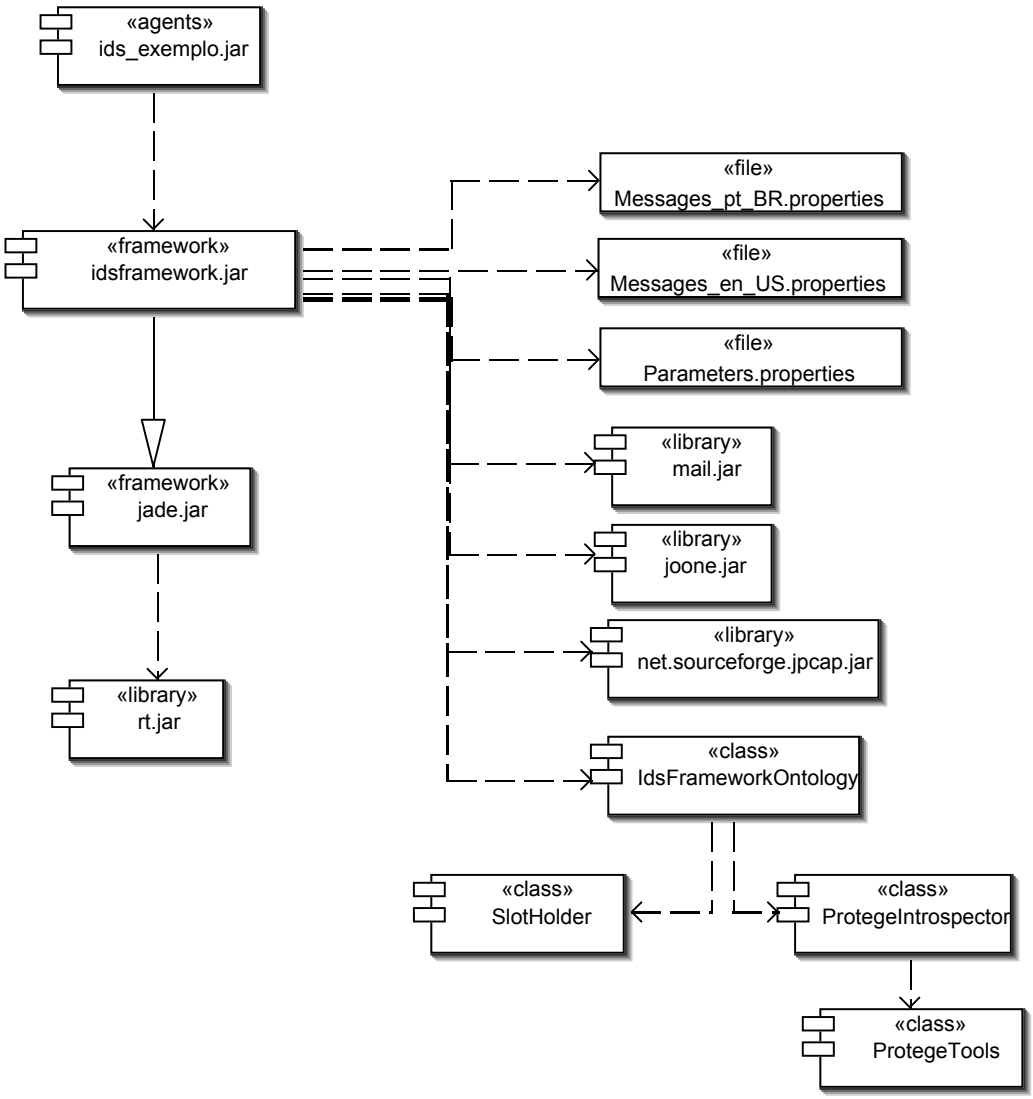


Figura 4.23 - Configuração física do *framework*

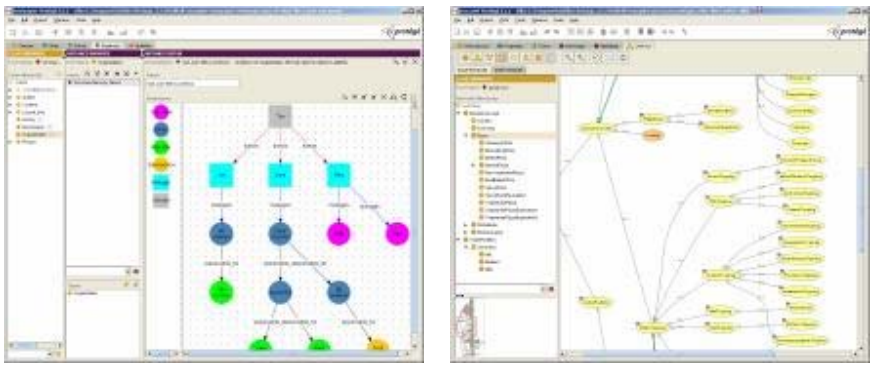
4.8 - COMPONENTES DE SERVIÇOS

Essa seção visa descrever alguns dos serviços disponíveis no *framework*, lembrando que mais serviços poderão ser adicionados nas futuras versões devido sua capacidade de extensão. Esses serviços deverão ser utilizados dentro das implementações dos agentes tornando maior a robustez de cada um. Aqui serão mostrados os seguintes serviços: suporte ao Protégé, rede neural artificial, capturador de pacotes e interface gráfica móvel.

4.8.1. Componente de interface com o Protégé

O Protégé é um editor para modelagem de ontologias e uma plataforma para construção de bases de conhecimento que possui licença de software livre e é totalmente baseado em Java. É disponibilizado em seu site oficial [5], juntamente com o código-fonte, documentações, *plugins* e tutoriais.

Pode ser operado por meio de dois editores, conforme mostrado na Figura 4.24: o Protege-Frames (a) e Protégé-OWL (b).



(a) (b)

Figura 4.24 - Editores do Protégé: Protégé-Frames e Protégé-OWL

O Protégé-Frames fornece uma interface que permite aos usuários a modelagem e construção das ontologias de domínios diversos. Implementa um modelo de conhecimento compatível com o protocolo da OKBC (Open Knowledge Base Connectivity) que representa uma ontologia em termos de classes dentro de uma hierarquia de conceitos, propriedades e relacionamentos do domínio, e um conjunto de instâncias dessas classes que são exemplares com as propriedades valoradas. Essa manipulação se assemelha bastante com os conceitos de orientação a objetos.

O Protégé-OWL é uma extensão que permite o suporte à OWL (Web Ontology Language). A OWL é o mais novo padrão para linguagens de ontologias da W3C (World Wide Web Consortium) para promover uma visão semântica da Web. Segundo a W3C [47], uma ontologia OWL deve incluir as descrições das classes, propriedades e suas respectivas instâncias.

O *framework* dá suporte às ontologias no Protégé por meio do objeto chamado Ids*Framework*Ontology em conjunto com o *plugin* `beangenerator` [48] que executa o processo de geração das classes em linguagem Java.

A Figura 4.25 mostra o modelo de classes que permite a manipulação de ontologias criadas no Protégé. Uma ontologia no Protégé consiste basicamente de:

- **Classes:** conceitos do domínio abordado que constituem uma hierarquia taxonômica;
- **Slots:** descrevem propriedades de classes e instâncias;
- **Facetas:** descrevem propriedades de *slots* e permitem a especificação de restrições nos valores dos *slots*;

A classe Ids*Framework*Ontology estende a classe Ontology do próprio JADE e implementa a interface ProtegeOntology que adiciona diversos serviços (ProtegeTools, ProtegeIntrospector e SlotHolder) para a conversão dos objetos definidos em Java para o contexto de uma mensagem no padrão FIPA-ACL.

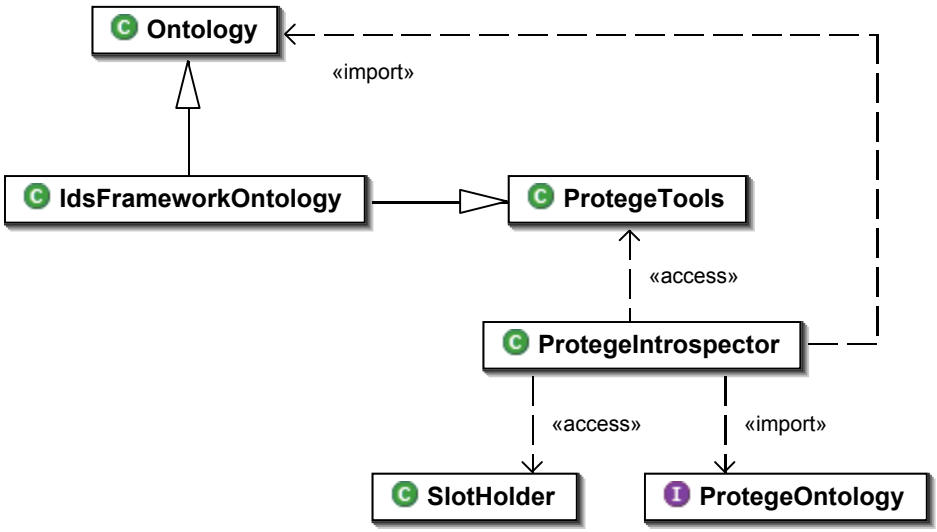


Figura 4.25 - Classes da ontologia do *framework*

Uma vez que os agentes trabalham com uma classe de ontologia composta pelos objetos modelados, toda comunicação é feita puramente com mensagens ACL em formato textual, inclusive o conteúdo da mensagem, mesmo que os conceitos e relações sejam objetos complexos. Isso é possível, pois o próprio JADE se encarrega pela codificação e decodificação dos conteúdos textuais em objetos.

4.8.2. Componente de Rede Neural Artificial

Para que os projetos dos SDIs possam implementar mecanismos simples de Inteligência Artificial, foi desenvolvido um componente de rede neural artificial chamado de SimpleNeuralNet que faz uso da biblioteca Joone [53] disponibilizado também sobre a licença de software livre.

O componente se baseia em uma rede neural do tipo MLP (Multi-layer Percéptron) que pode ser definida com diferentes topologias. São exigidas no mínimo 3 camadas - camada de entrada, camada escondida e camada de saída - que contenham ao menos um elemento processador cada uma. Por padrão, as conexões internas da rede neural proposta são feitas de forma que existam ligações entre todos os elementos de camadas subjacentes e a função de ativação é a Sigmoid para as camadas internas e para a camada de saída.

O modelo de classes do componente é mostrado na Figura 4.26. A classe `SimpleNeuralNet` estende a classe `NeuralNet`, originária do Joone, e fornece métodos específicos para que seja acoplada aos agentes. Além disso, implementa a interface `NeuralNetListener` que é responsável por avaliar os eventos emitidos em um treinamento.

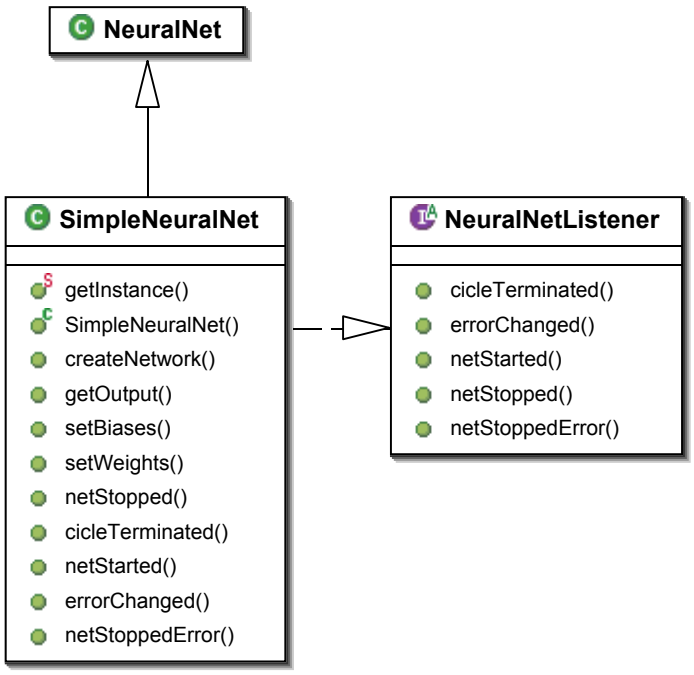


Figura 4.26 - Classes do componente de rede neural artificial

A utilização do componente é bastante simples, exigindo apenas a obtenção de uma instância da classe, configuração da topologia e ajustes nos pesos obtidos na fase de treinamento. A Figura 4.27 ilustra uma topologia possível para a rede neural artificial e a Tabela 4.3 demonstra o código necessário para construí-la usando o componente SimpleNeuralNet.

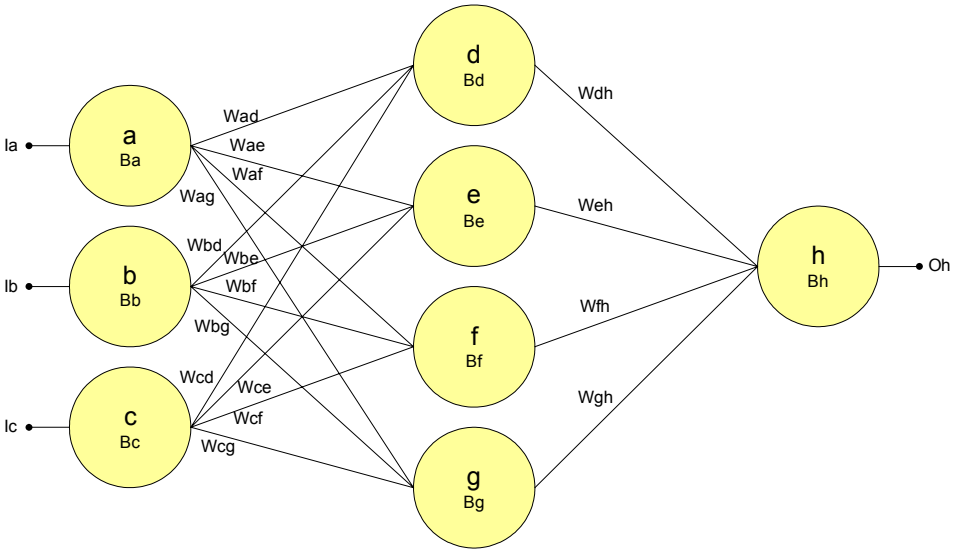


Figura 4.27 - Exemplo de rede neural artificial

Tabela 4.3 - Exemplo de código para o uso do SimpleNeuralNet

```
//Obtém a instância do SimpleNeuralNet
SimpleNeuralNet snn = SimpleNeuralNet.getInstance();

//Rotina de preenchimento de pesos e biases aleatórios
Random generator = new Random();
List weights = new ArrayList();
List biases = new ArrayList();
for (int i = 1; i++; i <= 15){
      if ( i >= 1 && i <= 8)
            biases.add(generator.nextDouble());
            weights.add(generator.nextDouble());
      }
}

//Configura o layout e os pesos
snn.setLayout({3,4,1});
snn.setBiases(ArrayList biases);
snn.setWeights(ArrayList weights);

//Cria uma entrada de teste
double[] input = new double[3] {0,0,1};

//Obtém a resposta da rede
double output = snn.getOutput(input);
```

Como pode ser observado, inicialmente foi obtida a instância do SimpleNeuralNet por meio do método `getInstance()`. Após isso, foram gerados valores randômicos para o preenchimento de um vetor de pesos e *biases*. Nesse ponto, vale ressaltar que a ordem de preenchimento dos vetores deve seguir a seguinte seqüência:

- **Vetor de pesos:** sinapses da esquerda para direita e de cima para baixo, sempre em relação aos neurônios mais a esquerda. No presente exemplo o vetor ficaria da seguinte forma: {*Wad, Wae, Waf, Wag, Wbd, Wbe, Wbf, Wbf, Wbg, Wcd, Wce, Wcf, Wcg, Wdh, Weh, Wfh e Wgh*};
- **Vetor de *biases*:** elementos da esquerda para direita e de cima para baixo. Exemplo: {*Ba, Bb, Bc, Bd, Be, Bf, Bg, Bh*}.

Por meio do método `setLayout()` pôde ser definida a topologia da rede neural artificial, seguindo o parâmetro passado que segue o padrão $\{N_1, N_2, ... , N_n\}$, onde N é o número de elementos em cada camada e *n* é o número total de camadas. Por exemplo, uma rede definida como {3, 10, 15, 7, 1} terá um total de cinco camadas, sendo que a camada de entrada terá 3 elementos, a primeira camada escondida terá 10 elementos, a segunda camada escondida terá 15 elementos, a última camada escondida terá 7 elementos e, finalmente, a camada de saída terá apenas 1 elemento.

Os pesos e *biases* foram atribuídos à rede e a mesma pôde ser interrogada utilizando o método `getOutput()` com os parâmetros de entrada adequados.

Outro ponto importante do componente, é que ele possibilita a reconfiguração da rede em tempo de execução dos agentes por meio dos métodos `setLayout()`, `setBiases()` e `setWeights()`, possibilitando que o conhecimento possa evoluir para solucionar novos problemas relativos a detecção de intrusão. Por exemplo, após novos treinamentos com métodos de ataques mais sofisticados, poderá surgir uma topologia e pesos mais apropriados que poderão ser diretamente passados para os agentes que fazem uso do componente.

4.8.3. Capturador de pacotes

O componente de captura de pacotes é sem dúvida um dos mais essenciais presentes no *framework*, pois é um serviço que viabiliza a operação dos agentes de coleta de dados da rede. Ele pode implementar tanto a captura do tráfego completo por meio da ativação do modo promíscuo da interface de rede, quanto a captura de pacotes apenas do *host* no qual o agente reside.

Diferentemente da maioria dos componentes presentes no *framework*, esse não pôde ser implementado integralmente em Java devido à necessidade de acessar nativamente a interface de rede. Dessa forma existirá uma restrição quanto à portabilidade dos agentes que o utilizam.

A solução foi construída sobre a biblioteca Jpcap [49] que permite que as aplicações Java capturem e enviem pacotes pela rede. O Jpcap faz o acesso nativo utilizando as conhecidas bibliotecas libpcap [50] e winpcap [51] que já são usadas em diversas ferramentas de segurança, como por exemplo, o Ethereal, Nmap e o Snort, e possibilita o trabalho com os seguintes tipos de pacotes: Ethernet, IPv4, Ipv6, ARP/RARP, TCP, UDP e ICMPv4. Além disso, possui licença de software livre LGPL, podendo operar nas seguintes plataformas: FreeBSD 3.x, Linux RedHat 6.1, Fedora Core 4, Solaris, e Microsoft Windows 2000/XP.

A arquitetura geral do componente é mostrada na Figura 4.28. A classe `Capture` representa uma captura e gera uma lista de pacotes (`Packet`), além disso, implementa a interface `PacketListener` por meio do método `packetArrived()` que atua como um sensor de novos pacotes.

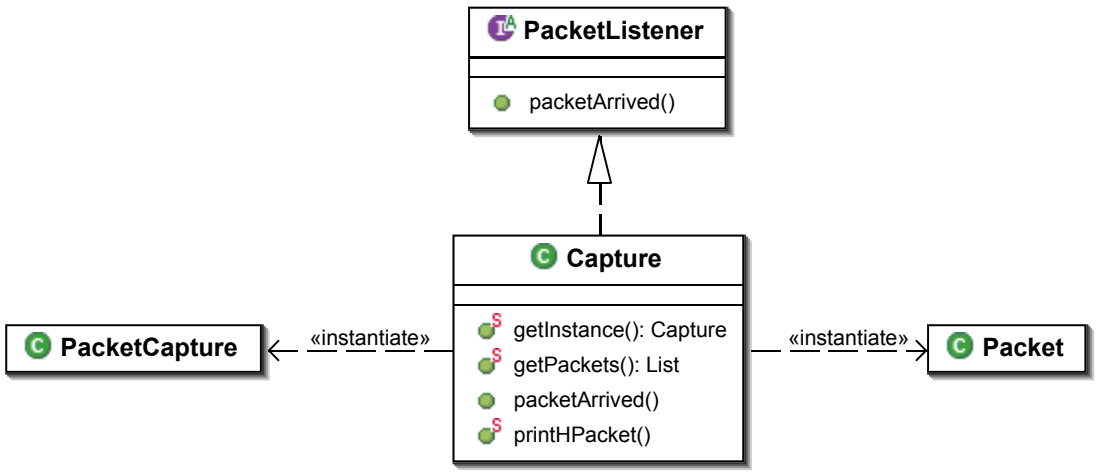


Figura 4.28 – Classes do componente de captura de pacotes

Seu uso se resume a chamar o método `getPackets()` passando como argumentos a interface de rede e o número desejado de amostras. No exemplo abaixo, serão coletados 1000 pacotes, objetos da classe `Packet`, pela interface `eth0`, armazenando-os em uma coleção de objetos chamada de `packets`.

```
List packets = Capture.getPackets("eth0",1000);
```

A classe Packet mostrada na Figura 4.29 representa um pacote coletado. Os campos foram organizados dessa forma para atender a necessidade de reconhecimento de padrões por meio do treinamento da rede neural.

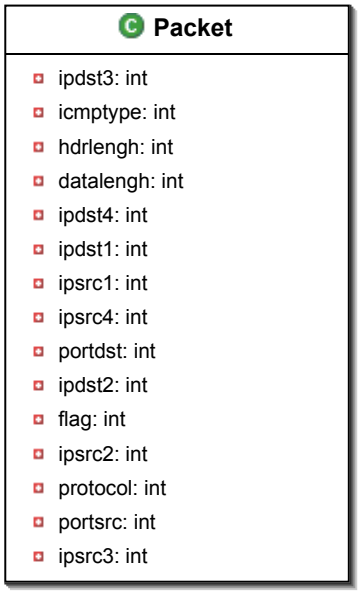


Figura 4.29 - Classe Packet

Os campos da classe Packet podem ser modificados pelos seus respectivos métodos acessores (*getters* e *setters*). O significado de cada campo é descrito na Tabela 4.4 e pode ser melhor esclarecido na revisão do TCP/IP presente no Apêndice A.

Tabela 4.4 - Atributos da classe Packet

Nome	Descrição	Nome	Descrição
ipdst1	1ª parte do endereço IP de destino (decimal)	***ipdst2***	2ª parte do endereço IP de destino (decimal)
ipdst3	3ª parte do endereço IP de destino (decimal)	***ipdst4***	4ª parte do endereço IP de destino (decimal)
ipsrc1	1ª parte do endereço IP de origem (decimal)	***ipsrc2***	2ª parte do endereço IP de origem (decimal)
ipsrc3	3ª parte do endereço IP de origem (decimal)	***ipsrc4***	4ª parte do endereço IP de origem (decimal)
portdst	Porta TCP/UDP de destino	***portsrc***	Porta TCP/UDP de origem
protocol	Numero do protocolo TCP/UDP	***icmptype***	Tipo de mensagem ICMP
flag	Flags do cabeçalho IP	***hdrlength***	Tamanho do cabeçalho IP
datalength	Tamanho da parte de dados do pacote IP		

4.8.4. Interface gráfica móvel

A interface gráfica móvel é um componente interessante para fins de depuração da mobilidade dos agentes e para servir de interface de diálogo com o usuário dos *hosts* visitados. Está presente em qualquer agente por meio do objeto BasicGUI, visto que todo agente herda as propriedades da classe GuiAgent.

Esse componente tem a propriedade de mostrar visualmente o agente em seu local de residência, levando consigo todos os dados atribuídos à interface. A Figura 4.30 mostra o aspecto de uma interface básica para o agente.

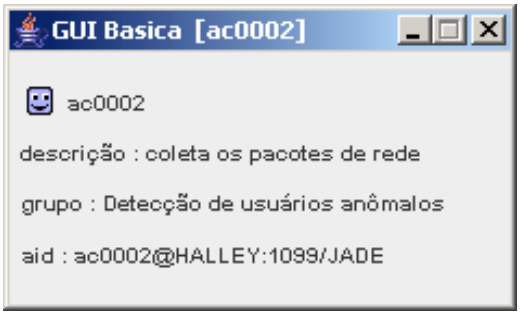


Figura 4.30 - Exemplo de interface gráfica de um agente

Para ativar o componente de interface gráfica do agente, deve ser invocado o método activateBasicGui()e deve ser implementada a interface BasicGuiInterface, procedimentos já executados por um BasicAgent. A interface BasicGuiInterface exige a definição de alguns métodos necessários ao componente de interface gráfica que configuram o nome, tipo, grupo, descrição dos agentes.

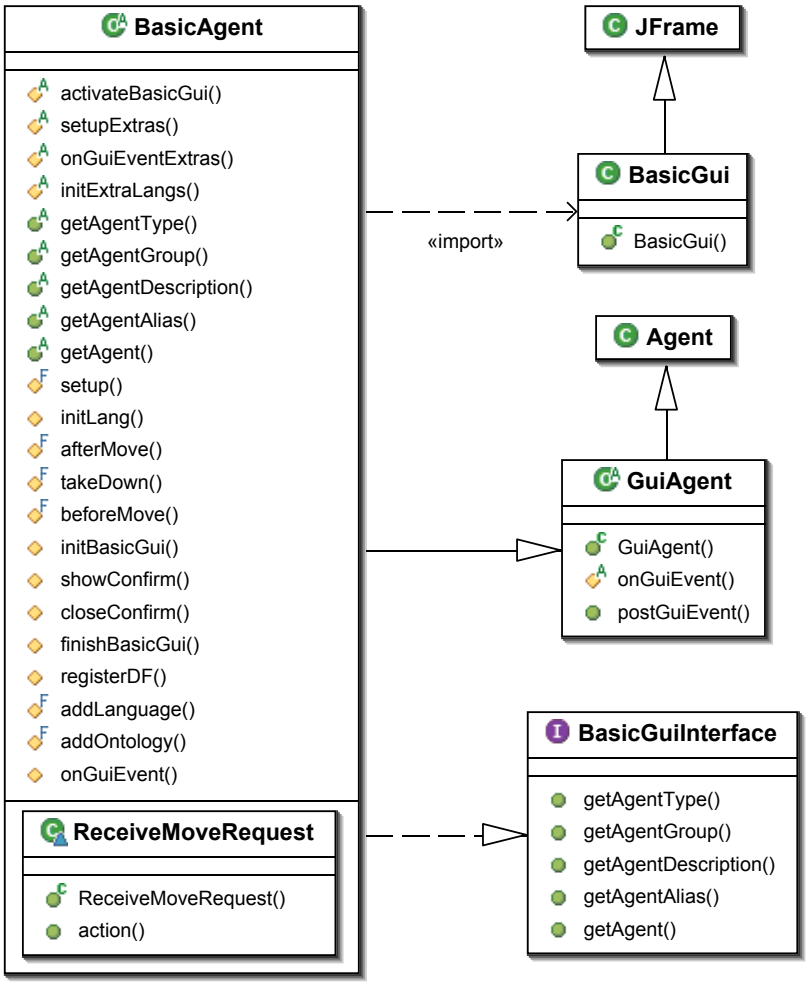


Figura 4.31 - Classe BasicGui

O componente também suporta o tratamento de eventos emitidos pelo usuário do sistema por meio do método `onGuiEvent()`. Dessa forma os agentes podem interagir visualmente com os usuários e responder às suas ações.

4.8.5. Utilitários

Além das bibliotecas voltadas para o próprio mecanismo de detecção, o *framework* disponibiliza algumas bibliotecas utilitárias que resolvem os problemas mais comuns envolvendo a plataforma de agentes e a comunicação dentro de um grupo de agentes. A Tabela 4.5 apresenta os principais utilitários.

Tabela 4.5 - Biblioteca de utilitários do *framework*

Nome	**Classe**	**Função**
`decodeIP()`	Utils	Extrai parte de um endereço IP
`getMessage()`	Utils	Recupera um campo existente no arquivo de mensagens (Message.properties) levando em conta a internacionalização
`getParameter()`	Utils	Recupera um campo do arquivo de configuração (Parameters.properties)
`printMessage()`	Utils	Registra as ações dos agentes levando em conta a internacionalização
`runSystemCommand()`	Utils	Executa um comando no sistema
`deregisterDF()`	HandleDF	Descadastra um agente do DF
`registerDF()`	HandleDF	Cadastra um agente no DF
`searchDF()`	HandleDF	Localiza um agente no DF
`informAgent()`	HandleMessage	Transmite uma informação a um agente
`listContainers()`	HandleMessage	Lista os containeres do macroambiente
`moveAgent()`	HandleMessage	Solicita a movimentação de um agente
`receiveResultFrom()`	HandleMessage	Recebe um resultado da ação de um agente específico
`requestAction()`	HandleMessage	Solicita uma ação a um agente
`searchLocation()`	HandleMessage	Localiza em qual container está um agente
`waitFor()`	HandleMessage	Paralisa todas as ações de um agente durante um tempo especificado

5 - IMPLEMENTAÇÃO E VALIDAÇÃO DE CENÁRIOS

O presente capítulo visa demonstrar a aplicabilidade do *framework* no desenvolvimento de um pequeno sistema de detecção de intrusão especializado em dois cenários de ataques e, também, visa avaliar os resultados obtidos. Lembrando que o foco desse trabalho se concentrou nos ataques internos.

Primeiramente, foi necessário propor um ambiente de validação que simulasse esses cenários e permitisse a coleta dos resultados. Após isso, foi necessário conceber, implementar e implantar os grupos e seus respectivos agentes especializados utilizando a metodologia proposta no trabalho. Por fim, foram realizados os ataques necessários e observados os comportamentos dos agentes para anotação dos resultados.

É esperada, com a leitura desse capítulo, uma melhor compreensão prática dos objetivos e benefícios do *framework* na construção de sistemas de detecção de intrusão baseados em agentes de software. Também que o mesmo sirva de tutorial para a criação de novos cenários, promovendo, assim, a evolução desse trabalho.

5.1 - AMBIENTE DE VALIDAÇÃO DOS CENÁRIOS

O ambiente proposto para desenvolvimento dos cenários, simulou uma pequena rede corporativa que foi segmentada em dois âmbitos: a Área de Produção, aonde residem os servidores da corporação, e a Área de Operação, aonde os funcionários e contratados utilizam suas estações de trabalho para exercerem suas atividades acessando os servidores de produção, conforme mostrado na Figura 5.1.

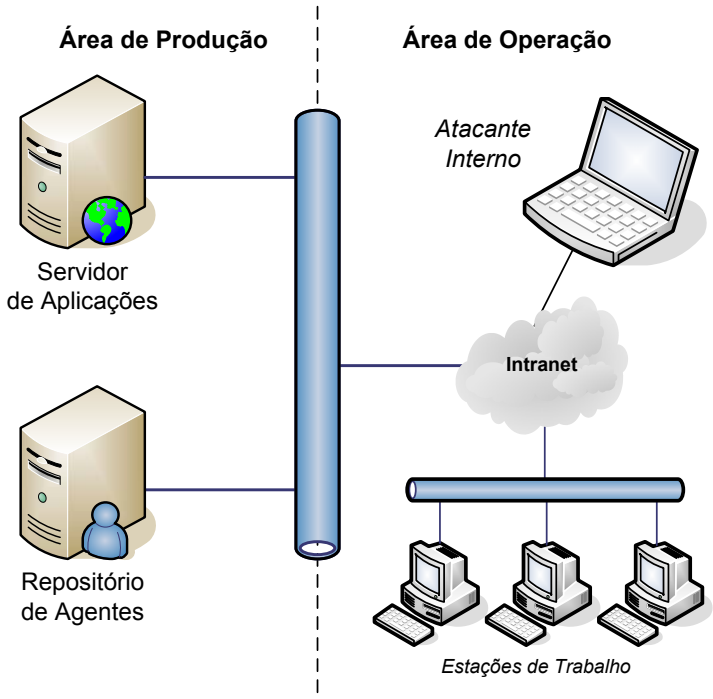


Figura 5.1 - Ambiente de rede das validações

Para o estudo do comportamento dos grupos de agentes mediante a alguns ataques, um novo componente foi inserido nesse ambiente – o atacante interno. Por exemplo, esse atacante poderia ser um contratado temporário que visaria roubar informações para posterior venda às empresas concorrentes. Ele faria isso utilizando a técnica de *port scanning* para o mapeamento dos serviços da área de produção, a fim de localizar alguma vulnerabilidade que permitisse sua entrada no servidor de aplicações.

Para proteger a rede contra essas ações, a abordagem apresentada foi dividida em dois cenários de detecção distintos - detecção de *port scanning* e detecção de presença de usuários anômalos – que acarretaram, respectivamente, na implementação dos seguintes grupos de agentes:

- **Grupo de detecção de *port scanning*:** Responsável pelo reconhecimento dessa atividade e, em caso positivo, geraria alertas por e-mail. Teria como ponto de partida o envio do seu agente de coleta para capturar pacotes diretamente no servidor de aplicações, conforme mostrado na Figura 5.2. Em caso de incidentes, enviaria também seu agente de reação.

- **Grupo de detecção de usuários anômalos:** Analisaria cada usuário que se conectasse ao servidor de aplicações e em caso de usuário anômalo ou inválido faria a expulsão automatizada do mesmo. Da mesma forma, faria o envio do seu agente de coleta e de reação para o servidor de aplicações.

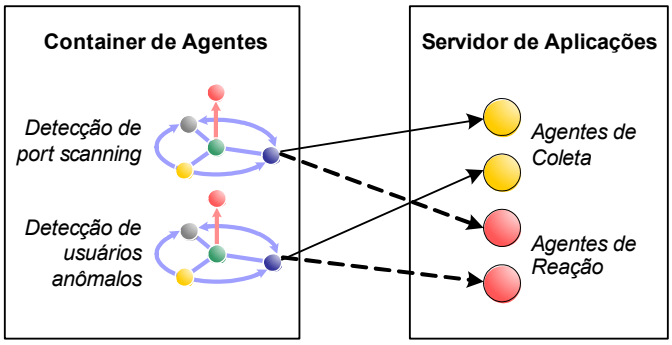


Figura 5.2 - Distribuição dos grupos de agentes

A próxima seção discorrerá sobre a metodologia de concepção e implementação que foram utilizadas na construção do grupos de agentes, iniciando-se pela modelagem da ontologia de comunicação que é um dos quesitos para construção dos agentes.

5.2 - CONCEPÇÃO E IMPLEMENTAÇÃO DOS GRUPOS DE AGENTES

5.2.1. Implementação da ontologia de comunicação

Os agentes precisam se comunicar por meio de um vocabulário formal e consensual para atingirem os objetivos do grupo. Para isso, o *framework* se baseia em ontologias que devem ser modeladas e implementadas previamente para cada cenário de detecção, contendo os termos básicos e suas relações.

Adicionalmente, o *framework* estende a ontologia básica da JADE (`jade.content.onto.Ontology`), o que permite tirar proveito do serviço de codificação de objetos em mensagens ACL e vice-e-versa, tornando a comunicação mais robusta e íntegra.

Para modelagem da ontologia foi utilizado o *software* Protégé [5] juntamente com o *plugin beangenerator* [48] que faz a geração das classes em linguagem Java. Os procedimentos para a modelagem da ontologia foram:

a) Abertura do projeto `IdsFrameworkOntology.pprj` que contém a modelagem de ontologias do *framework*. Esse projeto de ontologia se localiza no diretório `idsframework/protege`;
b) Criação dos novos conceitos, fazendo subclasses da classe `Concept`. Por exemplo, a Figura 5.3 mostra a criação dos conceitos `Collect` e `Packet`, para uma coleta e um pacote, respectivamente;

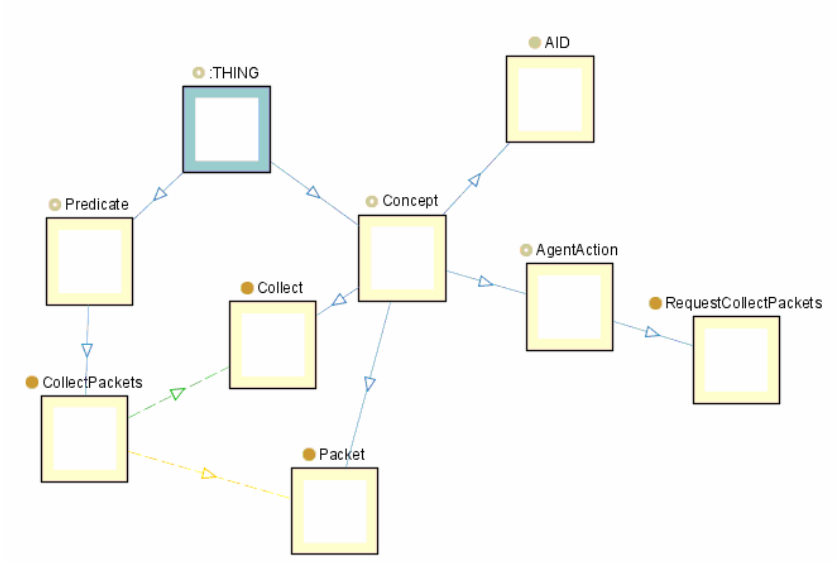


Figura 5.3 - Exemplo de modelagem no Protégé

c) Criação dos predicados, que são afirmações envolvendo mais de um conceito, por meio da criação de subclasses da classe `Predicate`. Como pode ser observado na Figura 5.3, foi criado um predicado `CollectPackets` que envolve os conceitos `Collect` e `Packets`, ou seja, esse predicado traz a representação de uma dada coleta de pacotes;
d) Criação das ações dos agentes com a definição de subclasses do conceito `AgentAction`. Conforme a Figura 5.3, foi criada a ação `RequestCollect` para representar uma solicitação de coleta de pacotes;

e) Para cada classe foi necessário definir os *slots*, ou seja, os atributos da classe, e seus respectivos tipos e cardinalidades. Como exemplo, a Figura 5.4 mostra os *slots* definidos para a classe `RequestCollect`.

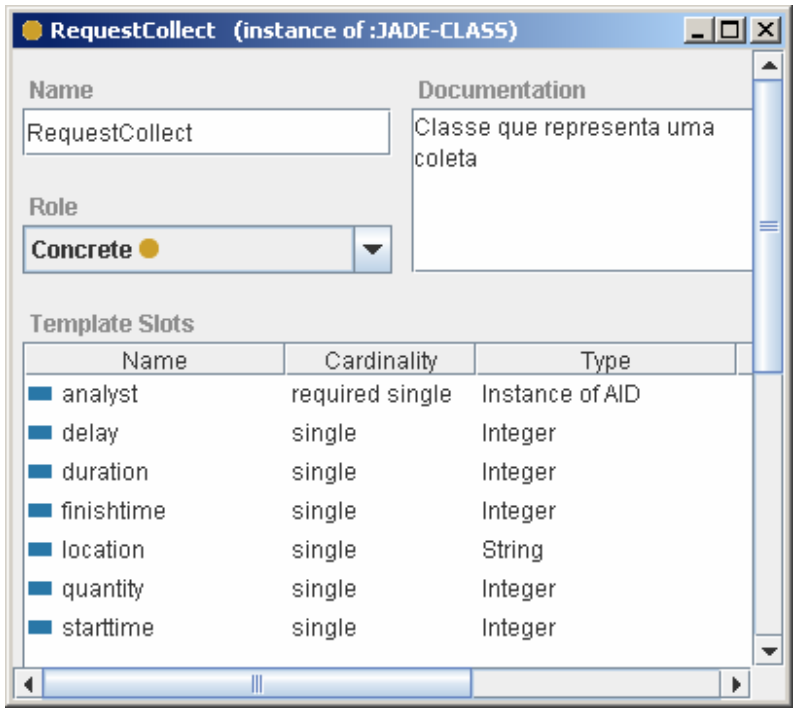


Figura 5.4 - Definição dos *slots* da classe

O resultado final da modelagem para os dois cenários é apresentado em forma de diagrama de classes pela Figura 5.5. Esse diagrama foi obtido por meio do *plugin Jambalaya* que faz parte da distribuição do Protégé.

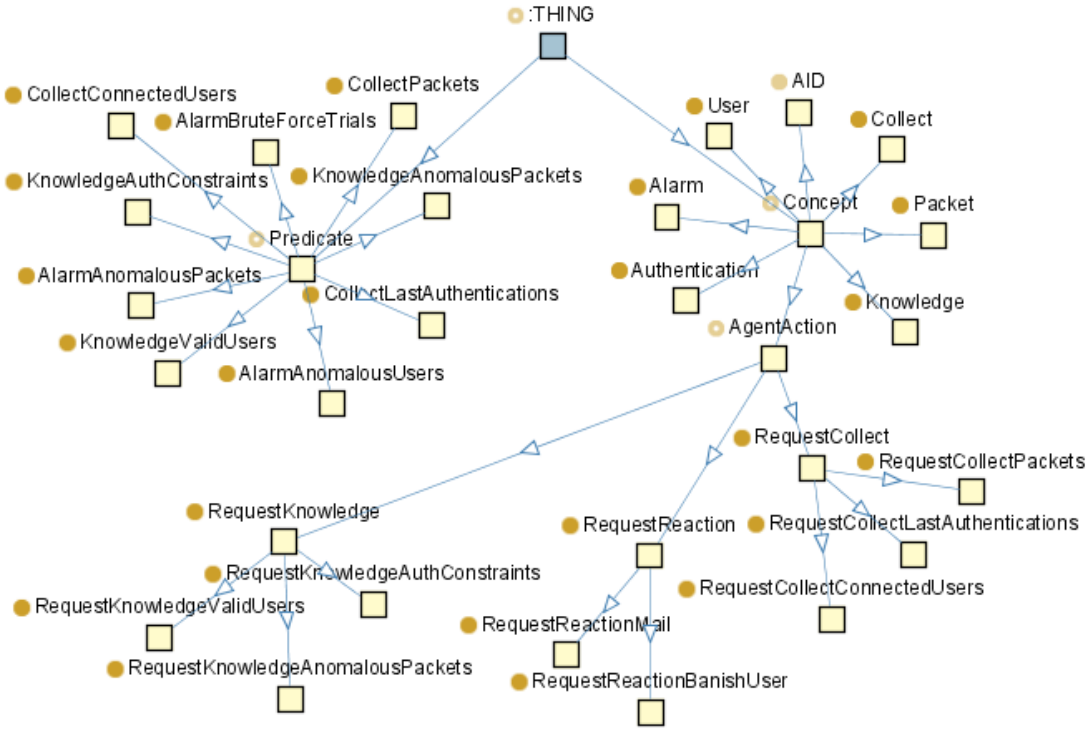


Figura 5.5 - Ontologia para comunicação dos agentes

Uma listagem mais detalhada das classes desenvolvidas para os dois cenários e suas respectivas funções pode ser observada na Tabela 5.1.

Tabela 5.1 - Classes do modelo da ontologia

Classe (Superclasse)	**Função**
`User (Concept)`	Abstração de um usuário com nome e identificador.
`Alarm (Concept)`	Abstração de um alarme com data, descrição, nível e localidade.
`Knowledge (Concept)`	Abstração de um conhecimento em uma data específica.
`Packet (Concept)`	Abstração de um pacote de rede contendo os campos do cabeçalho TCP/IP e dos protocolos TCP, UDP e ICMP.
`Collect (Concept)`	Abstração de uma coleta com data e localidade.
`AgentAction (Concept)`	Abstração de uma ação dos agentes.
`RequestKnowledge (AgentAction)`	Ação de solicitar renovação de conhecimento.
`RequestCollect (AgentAction)`	Ação de solicitar uma coleta informando o agente analista, o tempo entre coletas, a duração total, a hora de inicio e finalização, a quantidade de amostras e a localidade.

RequestReaction (AgentAction)	Ação de solicitar reação em uma dada localidade.
RequestKnowledgeValidUsers (RequestKnowledge)	Solicitação de conhecimento dos usuários válidos.
RequestKnowledgeAnomalousPackets (RequestKnowledge)	Solicitação de conhecimento dos padrões de pacotes anômalos.
RequestCollectConnectedUsers (RequestCollect)	Solicitação de coleta dos usuários conectados.
RequestCollectPackets (RequestCollect)	Solicitação de coleta de pacotes da rede.
RequestReactionMail (RequestReactiont)	Solicitação de envio de alarme por e-mail.
RequestReactionBanishUser (RequestReaction)	Solicitação de expulsão de usuário.
CollectPackets (Predicate)	Representa uma coleta com a lista de pacotes.
CollectConnectedUsers (Predicate)	Representa uma coleta com a lista de usuários conectados
AlarmAnomalousUsers (Predicate)	Representa um alarme de usuários anômalos
AlarmAnomalousPacket (Predicate)	Representa um alarme de pacotes anômalos
KnowledgeValidUsers (Predicate)	Representa o conhecimento sobre usuários válidos
KnowledgeAnomalousPackets (Predicate)	Representa o conhecimento sobre os padrões de pacotes anômalos.

Para exportar o modelo ontológico desenvolvido no Protégé de forma a ser compartilhado entre os agentes do *framework*, foram necessários os seguintes passos:

a) Por meio da aba *Ontology Bean Generator*, foram preenchidos os campos do formulário de exportação conforme a Figura 5.6. Para que a ontologia fosse registrada automaticamente nos agentes do *framework*, o nome do pacote foi especificado como `br.unb.idsframework.ontologies` e o domínio da ontologia como `idsframework`. O local de criação apontou para o diretório do projeto;

b) Após isso, as classes foram criadas por meio do comando *Generate Beans*.

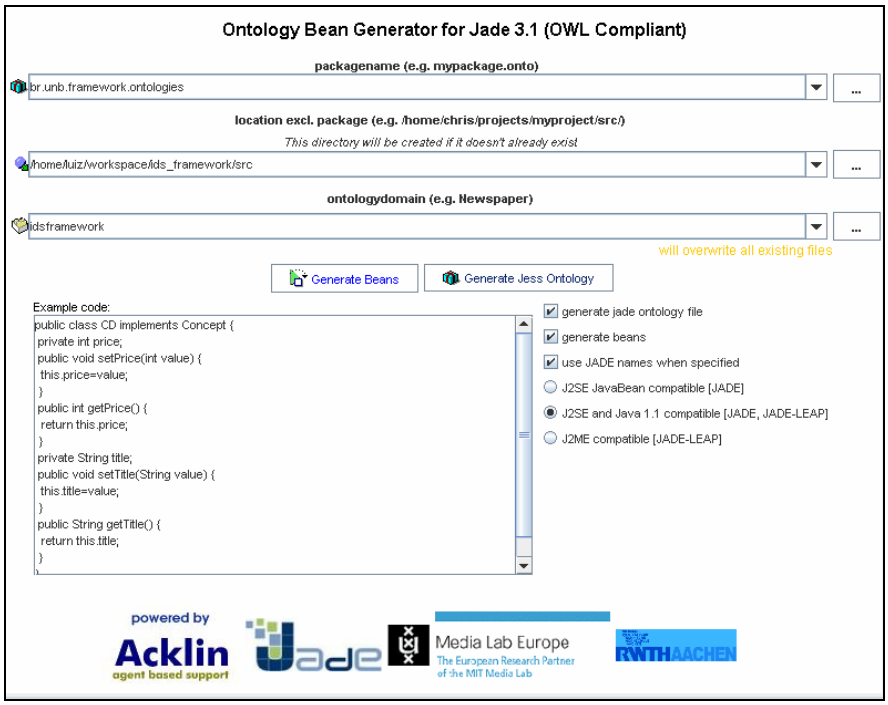


Figura 5.6 - Geração da ontologia com o *Beangenerator*

Os diversos elementos foram exportados como classes codificadas em linguagem Java dentro do pacote `br.unb.idsframework.ontologies`, incluindo a própria definição da ontologia presente na classe `IdsFrameworkOntology`. A Tabela 5.2 mostra um exemplar de código gerado para a classe `RequestCollect`.

Tabela 5.2 - Código fonte da classe de requisição de coleta (`RequestCollect.java`)

```
package br.unb.idsframework.ontologies;

import jade.content.AgentAction;
import jade.core.AID;

public class RequestCollect implements AgentAction {

   private String location;
   private int finishtime;
   private int delay;
   private AID analyst;
   private int duration;
   private int starttime;
   private int quantity;

   public void setLocation(String value) {
    this.location=value;
   }
   public String getLocation() {
     return this.location;
   }
   public void setFinishtime(int value) {
    this.finishtime=value;
   }
   public int getFinishtime() {
     return this.finishtime;
   }
            .
            .
            .

   public void setQuantity(int value) {
    this.quantity=value;
   }
   public int getQuantity() {
     return this.quantity;
   }
}
```

Pode ser observado que a classe `RequestCollect` faz todo o mapeamento dos *slots* em termos de atributos privados (`finishtime`, `delay`, `analyst`, `duration`, `starttime` e `quantity`) e oferece os respectivos métodos acessores (*getters* e *setters*) que manipulam seus valores. Tipos de dados mais complexos também podem ser exprimidos, como o caso do atributo `analyst` que é uma instância de AID (*Agent Identifier*).

Para exemplificar o uso da ontologia desenvolvida, podem ser consideradas as seguintes situações:

- O agente de coordenação inicia o processo de monitoração de pacotes enviando uma mensagem ACL para o agente de coleta contendo a requisição, ou seja, o conceito `RequestCollect`;
- O agente de coleta começa a colher informações no servidor e as envia ao agente de análise por meio de uma mensagem ACL cujo conteúdo é uma lista de pacotes capturados representada pelo predicado `CollectPackets`;
- Após perceber que alguns pacotes são suspeitos, o agente de análise emite uma mensagem ACL de alarme usando o predicado `AlarmAnomalousUser` e assim por diante.

A Tabela 5.3 demonstra a mensagem ACL gerada para uma requisição de coleta. Nesse exemplo, o agente de coordenação `ao1001@halley` enviou uma mensagem ao agente de coleta `ac0001@halley` solicitando uma coleta no container `appserver`, a cada intervalo de 1000 ms, sem agendamento de início ou finalização e em uma quantidade de 1000 amostras por coleta que devem ser entregues ao agente de análise `aa2001@halley`. A ontologia da mensagem foi especificada como `idsframework`

Tabela 5.3 – Exemplo de mensagem ACL de requisição de coleta

```
(REQUEST
 :sender    ( agent-identifier
                        :name ao1001@HALLEY:1099/JADE
                        :addresses (sequence
                                          http://HALLEY:7778/acc
                                          http://HALLEY:2014/acc ))
 :receiver  (set
                  ( agent-identifier :name ac0001@HALLEY:1099/JADE ))
 :content   "((action
                  ( agent-identifier :name ac0001@HALLEY:1099/JADE)
                  ( RequestCollect
                        :location appserver
                        :finishtime 0
                        :delay 1000
                        :analyst
                              (agent-identifier :name
aa2001@HALLEY:1099/JADE)
                        :duration 0
                        :starttime 0
                        :quantity 1000)))"
 :language  fipa-sl  :ontology  idsframework )
```

Como afirmado anteriormente, o conteúdo da mensagem trafega no formato textual FIPA-ACL, sem elementos codificados ou serializados, mesmo que em sua origem se caracterizassem por objetos da ontologia.

5.2.2. Grupo de detecção de *port scanning*

Primeiramente, as tarefas do grupo de detecção de *port scanning* foram divididas entre seus agentes especializados que foram nomeados seguindo um padrão pré-definido. Nesse sentido, as responsabilidades foram designadas do seguinte modo:

- **Agente de coleta (ac0002):** responsável pela captura os pacotes que chegam à interface de rede do servidor;
- **Agente de armazenamento (aw3002):** responsável pelo armazenamento e divulgação dos padrões e assinaturas de *port scanning*;
- **Agente de análise (aa2002):** responsável pelo reconhecimento dos padrões de *port scanning* com o auxílio de uma rede neural artificial;
- **Agente de reação (ar4002):** responsável pelo envio de e-mail contendo os dados do *port scanning* para o administrador;
- **Agente de coordenação (ao1001):** responsável pelas solicitações das atividades de coleta e de reação aos agentes.

As seções seguintes abordam sobre a aplicabilidade das redes neurais artificiais no problema de reconhecimento de padrões de *port scanning* e a implementação de cada agente desse grupo em particular, dando enfoque às facilidades promovidas pelo *framework* bem como o uso dos seus componentes de serviços.

5.2.2.1. Concepção e treinamento da rede neural artificial MLP

Antes de proceder com a implementação do agente de análise usando o componente SimpleNeuralNet para o reconhecimento dos padrões de *port scanning*, foi necessário conceber uma topologia e as respectivas configurações (pesos, *biases*, funções de ativação e etc.) que representam o conhecimento adquirido. Para isso, foi necessário obter e formatar uma base de dados com esses ataques para o dimensionamento e treinamento da rede neural artificial.

Não sendo exatamente o foco desse trabalho o estudo e aplicação das metodologias para análise, extração e tratamento de bases de ataques, foram aproveitadas as experiências de outras pesquisas, como em [57], [59] e [58]. Dessa forma, foi obtida uma base de dados gerada a partir de um ambiente controlado que simulava atividades normais e maliciosas. A base de dados contava com um total de 5980 amostras, sendo 3075 identificadas como ataques de *port scanning* nas diferentes técnicas apresentadas no Capítulo 2 (TCP connect(), TCP SYN, etc.).

A Tabela 5.4 mostra um trecho dessa base de dados que possui 15 campos de entrada (C1 a C15) e um de saída (C16). Os campos de entrada são respectivamente: número do protocolo; 1º, 2º, 3º e 4º octeto do IP de origem; 1º, 2, 3º e 4º octeto do IP de destino, porta de origem, porta de destino, *flags* do TCP, tamanho do cabeçalho TCP/UDP, tamanho do *payload* TCP/UDP e tipo de mensagem ICMP. Já o campo de saída é o tipo da amostra que pode ser ataque ou não. Caso seja necessário, pode ser consultado o Apêndice A que aborda e define cada campo presente na pilha de protocolos TCP/IP.

Tabela 5.4 - Amostras da base de dados de ataques de port scanning

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
6	172	16	0	2	192	168	0	254	32974	80	24	20	21	0	**0**
6	172	16	0	2	192	168	0	254	32974	80	24	20	256	0	**0**
6	172	16	0	2	205	188	11	220	32777	5190	24	20	6	0	**0**
6	172	16	0	2	207	46	106	55	32778	1863	24	32	5	0	**0**
6	172	16	0	1	172	16	0	2	1	1671	20	20	0	0	**1**
6	172	16	0	1	172	16	0	2	1	1676	20	20	0	0	**1**
6	172	16	0	1	172	16	0	2	100	1203	20	20	0	0	**1**

***Legenda**: Número do protocolo (C1), 1º octeto do IP de origem (C2), 2º octeto do IP de origem (C3), 3º octeto do IP de origem (C4), 4º octeto do IP de origem (C5), 1º octeto do IP de destino (C6), 2º octeto do IP de destino (C7), 3º octeto do IP de destino (C8), 4º octeto do IP de destino (C9), porta de origem (C10), porta de destino (C11), flags do TCP (C12), tamanho do cabeçalho TCP/UDP (C13), tamanho do payload TCP/UDP (C14), tipo de mensagem ICMP (C15) e o tipo da amostra (C16).*

Vale ressaltar que a base de dados que viria a ser utilizada passou previamente por um processo de seleção. Devido ao fato das atividades de *port scanning* não envolverem os conteúdos das aplicações, só foi necessário extrair os dados de controle que estão presentes nos cabeçalhos do TCP, UDP e ICMP, ou seja, os dados abaixo da camada de aplicação do TCP/IP.

Além disso, foi feito o tratamento e normalização desses dados para que pudessem ser utilizados no treinamento da rede neural artificial que trabalha com faixas discretas. Esse processo seguiu as seguintes etapas:

a) Conversão de todos os dados para o valor decimal, pois originariamente se apresentavam em base binária, octal e hexadecimal;
b) Para um pacote do tipo TCP, foi atribuído o valor zero no campo de tipo de ICMP (C15), pois esse não existia;
c) Para um pacote do tipo ICMP, foi atribuído o valor 0 nos campos de portas (C10 e C11), *flags* (C12), tamanho do cabeçalho (C13) e do payload (C14);
d) Para um pacote do tipo UDP, foi atribuído o valor 0 no campo de *flags* do TCP (C12);
e) Para um pacote gerado em uma situação de *port scanning*, foi atribuído o valor 1 no campo de tipo de amostra (ataque), enquanto que em um pacote gerado em uma situação normal de uso foi atribuído o valor 0 (normal).

Seguindo as sugestões de treinamento presentes em [58], foi utilizado o *software* EasyNN [60] para simular e treinar uma rede neural artificial com o objetivo de obter as configurações que seriam aplicadas na SimpleNeuralNet. Primeiramente, os dados foram importados, como mostrado na Figura 5.7. Das 5980 amostras, 264 foram selecionadas para validação dos resultados, 36 para testes externos e 7 foram excluídas por estarem redundantes ou falhas.

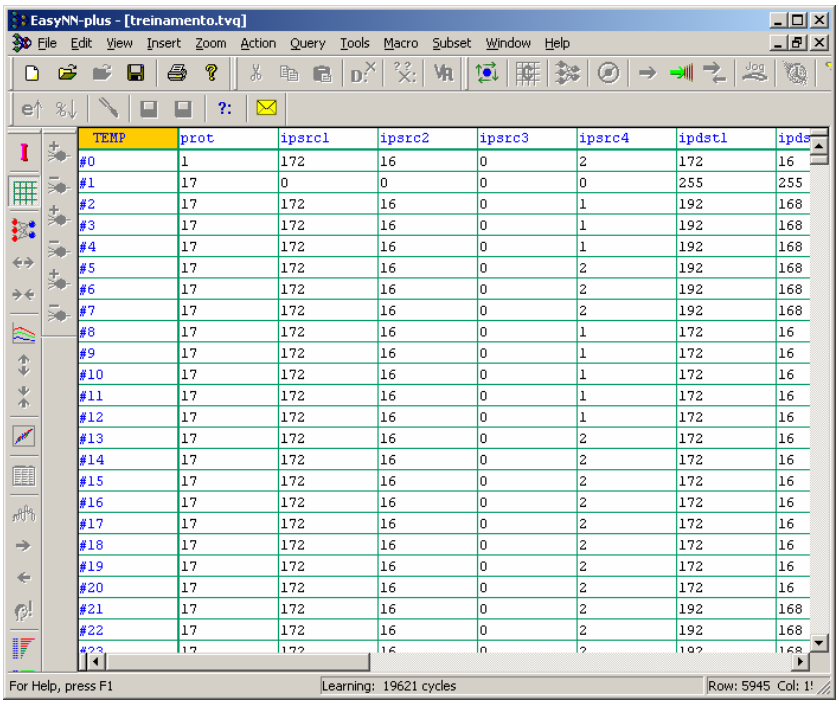


Figura 5.7 – Base de dados importada para o EasyNN

Após isso foram propostas e avaliadas algumas topologias para a rede, levando em conta sua capacidade de aprendizagem e simplicidade. Dentre elas, a topologia com três camadas, mostrada na Figura 5.8, foi eleita. A camada de entrada ficou com 15 elementos, a camada intermediária com 4 elementos e a camada de saída com apenas um elemento.

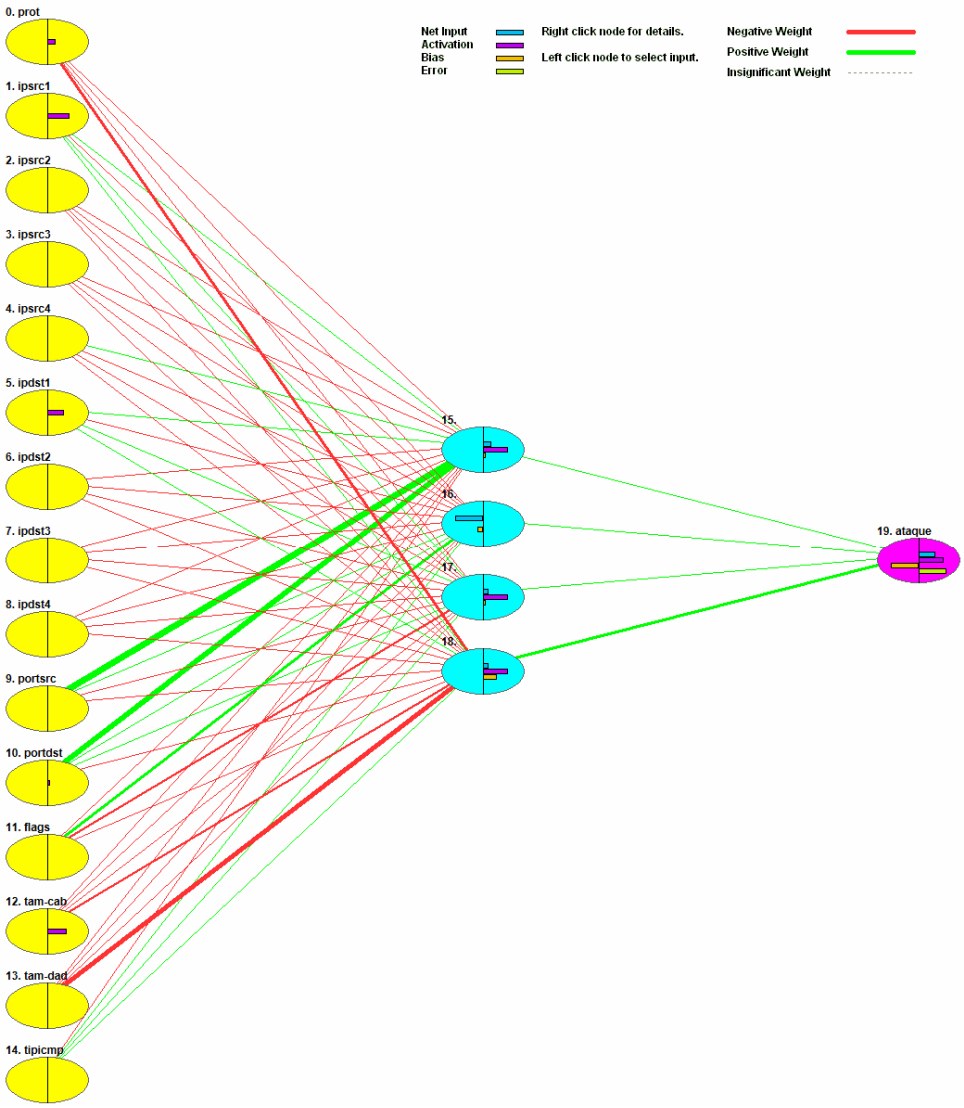


Figura 5.8 – Topologia da rede neural artificial gerada no EasyNN

Por padrão, o EasyNN opera com redes do tipo MLP com conexões entre todos os elementos de camadas subjacentes, sendo que são aplicadas funções de ativação *Sigmoid* apenas nos elementos intermediários e de saída. Sendo assim, os 15 elementos de entrada apenas serviram como sensores dos 15 campos de entrada presentes na base de dados de ataques.

O treinamento foi feito de forma supervisionada com o auxílio do algoritmo *backpropagation*. A taxa de aprendizagem foi ajustada para 0.60 e o *momentum* para 0.80. Os limiares do treinamento foram ajustados para atingirem erros inferiores a 0.05 e para garantir 100% de sucesso nos resultados das amostras de validação.

Após 64.928 ciclos o treinamento foi finalizado, pois os resultados já estavam satisfatórios para o reconhecimento dos padrões de *port scanning*, de acordo com a Tabela 5.5. A classificação das amostras de validação alcançou um nível de 99,6 % de acerto. O impeditivo para a conclusão automática do treinamento foi o máximo valor de erro no treinamento (0.952961) que ficou alto devido a existência de um número pequeno de amostras de difícil classificação. A finalização foi feita a fim de evitar que houvesse uma saturação no treinamento, o que diminuiria a capacidade de generalização da rede em relação a novas amostras.

Tabela 5.5 - Resultados do treinamento

Tipo do resultado	**Valor**
Número de ciclos	64928
Erro médio no treinamento	0.000989
Erro máximo no treinamento	0.952961
Erro médio na validação	0.003635
Desempenho da validação	99,6 %

A curva de aprendizagem durante o treinamento pode ser observada na Figura 5.9. O erro máximo, representado pela linha vermelha (curva superior), foi caindo lentamente, mas sempre com valores acima dos esperados. Já as taxas médias de erro de validação e de treinamento (curvas inferiores) se mantiveram abaixo do limiar de 0.005 em todo momento, sendo assim, para a maioria das amostras houve pouco erro na classificação dos padrões.

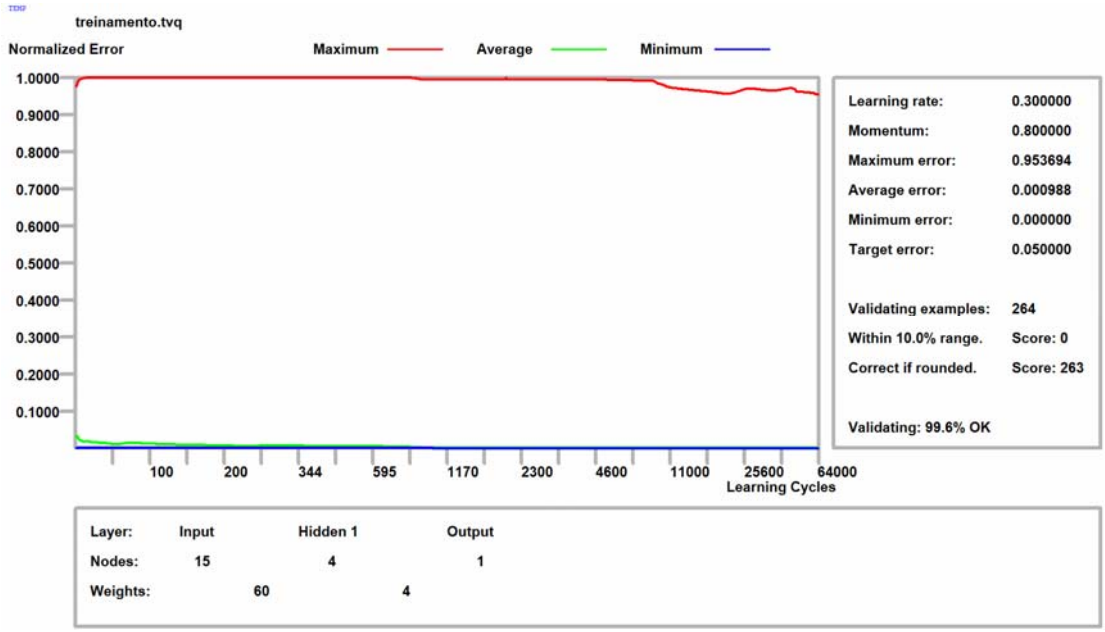


Figura 5.9 - Curva de aprendizagem dos padrões gerada no EasyNN

A ferramenta também disponibiliza um relatório que mostra quais amostras estão provocando os maiores erros na classificação de padrões. Conforme a Figura 5.10, pode ser confirmado que um número pequeno de amostras, no total de onze, estão sendo de difícil classificação.

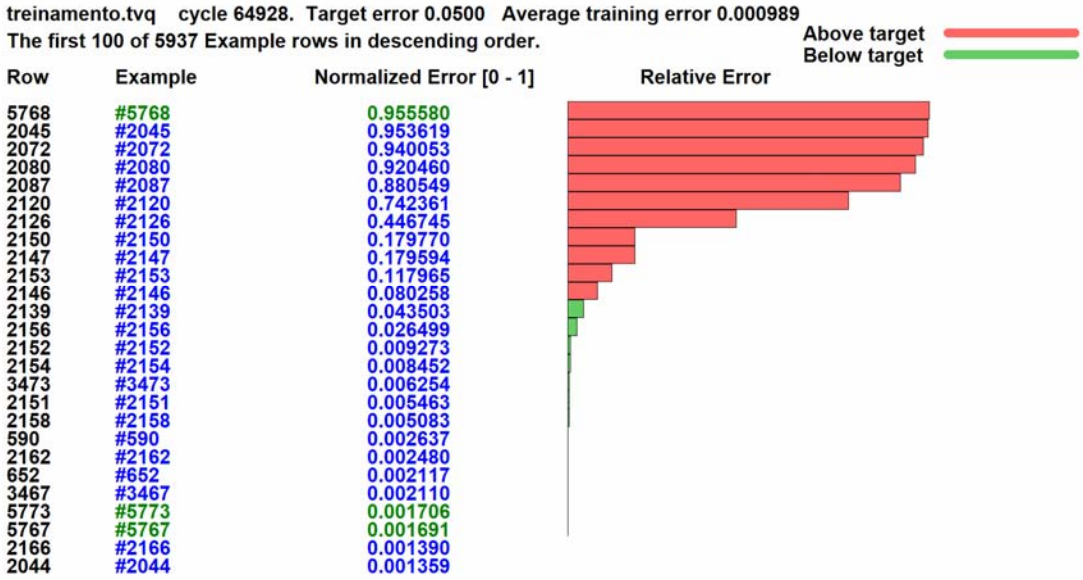


Figura 5.10 – Relatório do EasyNN com as amostras que mais causaram erros de classificação

Os campos que mais são relevantes para a classificação entre uma atividade normal e de *port scanning* também podem ser visualizados. De acordo com a Figura 5.11, esses campos são: porta de destino, porta de origem, tamanho do *payload*, *flags*, número do protocolo e tamanho do cabeçalho.

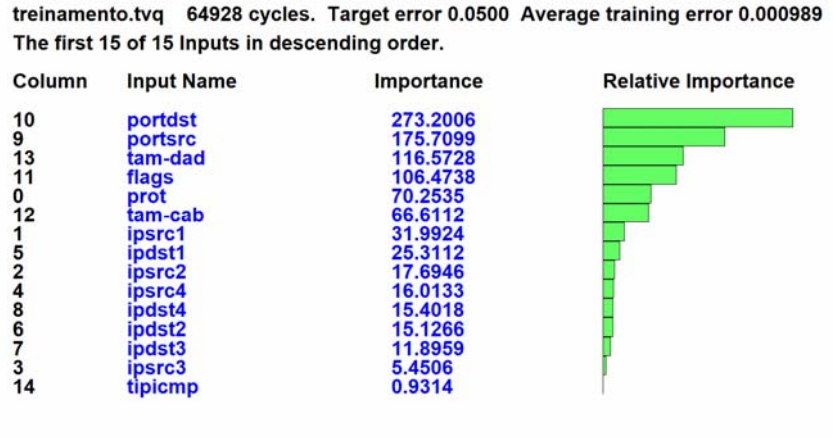


Figura 5.11 - Relatório do EasyNN com os campos mais relevantes para a classificação

Outra visualização interessante é o plano de classificação das amostras de acordo com a aprendizagem da rede neural artificial, mostrado na Figura 5.12. Como pode ser notada, a maior parte das amostras de validação (quadro da esquerda) e de treinamento (quadro da direita) se posicionaram acima ou abaixo do limiar de classificação. Porém, algumas amostras se posicionaram na fronteira de classificação, gerando os erros de classificação.

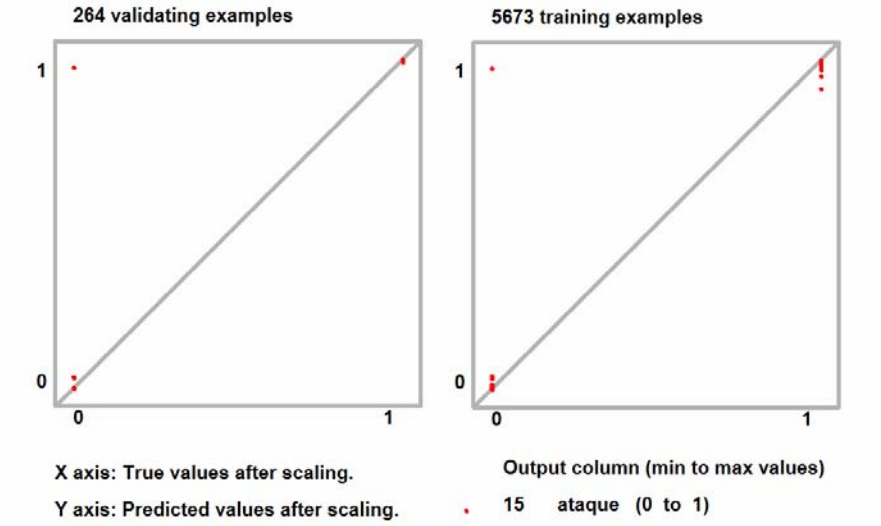


Figura 5.12 - Plano de classificação das amostras gerado pelo EasyNN

Por fim, foram obtidos os pesos das conexões presentes na rede neural artificial treinada, lembrando que os pesos sinápticos representam o próprio conhecimento adquirido. A Tabela 5.6 relaciona os pesos sinápticos e *bias* ajustados ao final do treinamento. Esses foram guardados no agente de armazenamento para que compusessem a rede neural artificial do agente de análise de *port scanning*.

Tabela 5.6 - Pesos sinápticos e *bias* encontrados para a rede neural artificial

Elementos Entrada e Saída *(Bias)*	**Elementos Intermediários *(Bias)***			
	Elemento 15 *(-34.176629)*	**Elemento 16** *(5.194010)*	**Elemento 17** *(-14.518896)*	**Elemento 18** *(12.5623668)*
Elemento 0	47.585167	-13.347998	-6.391445	-5.808830
Elemento 1	-23.738714	-5.134728	4.156246	7.662771
Elemento 2	6.992793	-2.556115	-2.313982	-2.655102
Elemento 3	3.854541	-0.484093	-0.198829	-1.079797
Elemento 4	10.080295	-9.361393	-2.599433	-2.454119
Elemento 5	-15.625877	-5.392606	3.274742	6.105258
Elemento 6	-15.625877	-5.392606	3.274742	6.105258
Elemento 7	18.736274	-7.099260	-2.521504	-2.254343
Elemento 8	6.207129	-1.135526	-1.239493	-1.265290
Elemento 9	6.306950	1.311484	-2.681825	-2.763621
Elemento 10	-0.115663	126.814498	94.936082	10.318969
Elemento 11	-16.252872	-13.298848	220.688525	1.598029
Elemento 12	18.516507	45.976831	-36.713115	-34.165496
Elemento 13	52.423896	-29.386275	-8.527528	6.507740
Elemento 14	84.769627	-38.918690	-8.762229	-3.947438
Elemento 19 *(-3.413491)*	-45.906944	25.155377	8.871492	10.737856

5.2.2.2. Implementação do agente de coleta

O primeiro passo para implementar qualquer agente de coleta é criar uma nova classe que estenda a classe `BasicColletor` e sobrescrever o método `collect` com os procedimentos necessários. O método `collect` é chamado ciclicamente e o seu resultado é enviado ao agente de análise que foi designado pelo agente de coordenação.

No caso do agente de coleta de pacotes, o principal objetivo foi capturar os pacotes que chegavam à interface de rede, transformando-os em um formato compreensível para os outros agentes. Isso pôde ser feito com uso da ontologia `idframework` registrada em todos agentes.

A Tabela 5.7 mostra o trecho principal do código desenvolvido para a classe `PacketCollector` que define o agente de coleta de pacotes. Para implementar o método `collect` foi utilizado o componente `Capture` que faz a captura dos pacotes endereçados ao servidor. Sua utilização é bastante simples, bastando para isso passar o identificador da interface de rede (`eth0`) e o número de amostras desejadas como parâmetros do método `getPackets`.

Tabela 5.7 – Trecho do código fonte do agente de coleta (PacketCollector.java)

```
public class PacketCollector extends BasicCollector {
                                        .
                                        .
    protected ContentElement collect(Agent agent, AID analyst){

        Collect collect = new Collect();
        collect.setDate( new Date() );
        collect.setLocation(agent.here().getName());
        List packets = Capture.getPackets("eth0", 1000);
        CollectPackets cp = new CollectPackets();
        cp.setPackets(packets);
        cp.setCollect(collect);
        return cp;

    }
                                        .
                                        .
}
```

Após o método `getPackets` ser invocado, o mesmo inicia o processo de captura de pacotes da interface até que se complete o número de amostras exigidas, gerando, assim uma coleção de objetos da classe `Packet`. Essa coleção de objetos foi agregada à classe `CollectPackets` juntamente com a instância da classe `Collect` para posterior envio ao agente de análise.

O método `collect` retorna um objeto do tipo `ContentElement` que é uma interface de abstração para qualquer conteúdo do tipo `Predicate`. Qualquer predicado implementa obrigatoriamente essa interface, incluindo a classe `CollectPackets`.

Após a coleta, os pacotes são encaminhados ao agente de análise. A próxima seção visa descrever a implementação desse agente.

5.2.2.3. Implementação do agente de armazenamento

Um agente de armazenamento deve ser criado a partir de uma extensão da classe básica `BasicWarehouse`, onde deve ser definido o método `query()`. Esse método é invocado sempre que houver uma solicitação de renovação de conhecimento por parte do agente de análise, processo que é transparente para o desenvolvedor.

A Tabela 5.8 mostra o trecho principal do código fonte do agente de armazenamento de padrões de *port scanning*. Nesse caso, a informação foi construída em cima da classe `KnowledgeAnomalousPackets` que é composta por uma instância de `Knowledge` em uma data específica e pelos parâmetros de configuração encontrados na fase de treinamento da uma rede neural artificial.

Tabela 5.8 - Trecho do código fonte do agente de armazenamento (PacketPatternWarehouse.java)

```
public class PacketPatternWarehouse extends BasicWarehouse {

    protected ContentElement query(){

        List nnweights = new ArrayList();
        List nnbiases = new ArrayList();

        //Pesos das conexoes das entradas com o neuronio 1
        nnweights.add(new Float(47.585167));
        nnweights.add(new Float(-13.347998));
        nnweights.add(new Float(-6.391445));
            .
            .
        //Bias dos neurônios de entrada
        nnbiases.add(new Float(0));
        nnbiases.add(new Float(0));
        nnbiases.add(new Float(0));
            .
            .
        Knowledge knowledge = new Knowledge();
        knowledge.setDate(20060404);
        KnowledgeAnomalousPackets kap = new
                              KnowledgeAnomalousPackets();
        kap.setNnabiases(nnbiases);
        kap.setNnaweights(nnweights);
        kap.setKnowledge(knowledge);

        return kap;
    }
}
```

Da mesma forma, o agente de armazenamento retornou um `ContentElement` com todas informações úteis para o agente de análise.

5.2.2.4. Implementação do agente de análise

Seguindo a mesma metodologia, o agente de análise se caracteriza por uma extensão da classe `BasicAnalyst` e definição do método `analyse` que deve retornar o resultado da avaliação ou detecção, propriamente dita. Para poder implementar os procedimentos de análise, o método `analyse` já fornece as referências para o objeto de coleta (`collect`) e também para o objeto de conhecimento (`knowledge`), ambos representados pela já citada interface `ContentElement`.

O agente de análise tem por objetivo reconhecer os padrões de *port scanning* e gerar um alarme em caso positivo. Para isso, foi utilizado o componente de rede neural artificial que foi inicializado com as configurações de topologia, pesos e *bias* obtidos na fase de treinamento.

A Tabela 5.9 demonstra o trecho do código fonte da classe `AnomalousPacketAnalyst` que define o agente de análise de *port scanning*. Para a implementação do método `analyse` foi utilizado o componente de rede neural artificial SimpleNeuraNet que foi inicializada com os pesos (`listKnowledgeWeights`) e *bias* (`listKnowledgeBiases`) recebidos do agente de armazenamento.

Tabela 5.9 – Trecho do código fonte do agente de análise (AnomalousPacketAnalyst.java)

```
public class AnomalousPacketAnalyst extends BasicAnalyst {

     protected ContentElement analyse(ContentElement collect,
ContentElement knowledge){

      double sumOutput = 0;
      int countPacket = 0;
      double alarmLevel = 0;

      CollectPackets cp = (CollectPackets) collect;
      KnowledgeAnomalousPackets kap = (KnowledgeAnomalousPackets)
knowledge;

      List listPackets = cp.getPackets();
      List listKnowledgeBiases = kap.getNnabiases();
      List listKnowledgeWeights = kap.getNnaweights();
```

```
      SimpleNeuralNet snn = SimpleNeuralNet.getInstance();
      snn.setBiases(listKnowledgeBiases);
      snn.setWeights(listKnowledgeWeights);

      Iterator itPackets = listPackets.iterator();

      while ( itPackets.hasNext() ){

            Packet packet = (Packet) itPackets.next();

            double[] input = new double[15];
            input[0] = packet.getProtocol();
            input[1] = packet.getIpsrc1();
            input[2] = packet.getIpsrc2();
            input[3] = packet.getIpsrc3();
            input[4] = packet.getIpsrc4();
            input[5] = packet.getIpdst1();
            input[6] = packet.getIpdst2();
            input[7] = packet.getIpdst3();
            input[8] = packet.getIpdst4();
            input[9] = packet.getPortsrc();
            input[10] = packet.getPortdst();
            input[11] = packet.getFlag();
            input[12] = packet.getHdrlengh();
            input[13] = packet.getDatalengh();
            input[14] = packet.getIcmptype();

            sumOutput += snn.getOutput(input);
            countPacket++;
      }

      alarmLevel = sumOutput/countPacket;

      System.out.println("ALARM LEVEL = " + alarmLevel);

      if (alarmLevel >= 0.9 ){

            Alarm al = new Alarm();
            al.setDate(new java.util.Date());
            al.setDescription("Pacotes maliciosos encontrados !");
            al.setLocation(cp.getCollect().getLocation());
            al.setLevel(new Float(alarmLevel).floatValue());

            AlarmAnomalousPackets aap = new AlarmAnomalousPackets();
            aap.setAlarm(al);
            aap.setPackets(listPackets);

            return aap;

      }
      else return null;
    }
                                    .
                                    .
}
```

O cálculo do nível de alarme geral de uma coleta foi determinado de acordo com a equação estatística da mediana (5.1), aonde L é o nível de alarme, N é o número de amostras, f é a função que calcula o nível de alarme de um dado pacote p_i. A função f pode ser considerada como a própria resposta da rede neural artificial ao receber os campos de cada pacote em suas entradas.

$$L = \frac{\sum_{i=1}^{N} f(p_i)}{N} \tag{5.1}$$

Foi estabelecido um limiar de 0.9 para que um alarme fosse emitido para o agente de coordenação. Esse limiar foi estabelecido empiricamente na decorrência do treinamento, levando em conta o erro máximo obtido.

5.2.2.5. Implementação do agente de reação

Por fim, o agente de reação segue a mesma linha dos anteriores por meio da extensão da classe `BasicReactor` e implementação do método `react`. O método `react` é invocado toda vez que chega uma mensagem de solicitação de reação (`RequestReaction`) do agente de coordenação.

Para fins de reuso dos agentes de coleta em outros grupos de detecção, também é passada a ação específica do agente que permite fazer uma coerção antecipada das ações, como foi o caso da ação `RequestActionMail`. A Tabela 5.10 mostra o trecho principal do código fonte criado para o agente de reação.

Tabela 5.10 - Trecho do código fonte do agente de reação (MailReactor.java)

```
public class MailReactor extends BasicReactor {

    protected void react(Concept action){

        if (action instanceof RequestReactionMail ){

                  RequestReactionMail rrm = (RequestReactionMail)
action;
                  Properties p = new Properties();
                  p.put("mail.host", "halley.local" );
                  Session session = Session.getInstance(p, null);
                  MimeMessage msg = new MimeMessage(session);

                  try {

                msg.setFrom(new
InternetAddress("ar4002@halley.local"));
                msg.setRecipient(Message.RecipientType.TO, new
InternetAddress("admin@halley.local"));
                msg.setSentDate(new Date());
                msg.setSubject( "Houve ataque em " +
rrm.getLocation());
                msg.setText(rrm.getMessage());

                Transport.send(msg);

                  }
            catch (AddressException e) {
                              .
                              .
                  }
            catch (MessagingException e) {
                              .
                              .
                  }
        }
                                    .
                                    .
}
```

Esse agente apenas fez uso da biblioteca `javax.mail` para fazer o envio de um e-mail por meio de um servidor SMTP contendo o container e a mensagem disparada pelo agente de análise.

5.2.2.6. Implementação do agente de coordenação

Para implementar o agente de coordenação, houve apenas a extensão da classe básica de agentes (`BasicAgent`) e definição de um roteiro fixo de inicialização dos cenários. Dessa forma, não foram definidos os métodos comuns a qualquer agente de coordenação.

A Tabela 5.11 mostra um trecho do código fonte desse agente. As definições foram feitas no método `setupExtras` que é chamado toda vez que um agente básico é criado. Basicamente, o agente de coordenação cria as solicitações de coleta com os parâmetros da coleta (intervalo, duração, numero de amostras e etc.) e as envia para o agente de coleta de pacotes e para o agente de coleta de usuários. Antes de cada solicitação, o agente de coordenação faz uma pausa para que o processo possa ser acompanhado.

Tabela 5.11 – Trecho do código fonte do agente de coordenação (MasterCoordinator.java)

```
public class MasterCoordinator extends BasicAgent {

    protected void setupExtras(){

        addBehaviour(new ReceiveAlarm(this));

        HandleMessage.waitfor(20000); //20 seconds

        //Cenário de deteccao de usuarios anômalos
        RequestCollect rc = new RequestCollect();
        rc.setAnalyst(new AID("aa2001",AID.ISLOCALNAME));
        rc.setDelay(1000);
        rc.setDuration(0);
        rc.setQuantity(1000);
        rc.setFinishtime(0);
        rc.setStarttime(0);
        rc.setLocation("appserver");
        HandleMessage.requestAction("ac0001",this,rc,
             IdsFrameworkOntology.getInstance(), new SLCodec());

        HandleMessage.waitfor(20000); //20 seconds

            //Cenário de deteccao de port scanning
            RequestCollect rc = new RequestCollect();
         rc.setAnalyst(new AID("aa2002",AID.ISLOCALNAME));
         rc.setDelay(2000);
         rc.setDuration(0);
         rc.setQuantity(1000);
         rc.setFinishtime(0);
         rc.setStarttime(0);
         rc.setLocation("appserver");
         HandleMessage.requestAction("ac0002",this,rc,
             IdsFrameworkOntology.getInstance(), new SLCodec());
    }
                                .
                                .
```

Como pode ser observado, não existiu muita autonomia nesse agente, pois ele apenas seguiu as diretrizes implementadas pelo desenvolvedor, o que contrastou com sua

proposta original. Isso se deve ao fato de que esse agente foi desenvolvido unicamente para testar os cenários de detecção e não para coordenar de forma inteligente os agentes em um ambiente de rede.

5.2.3. Grupo de detecção de usuários anômalos

O grupo de detecção de usuários anômalos seguiu as mesmas diretrizes de implementação do grupo anterior. Por isso são mostradas apenas as particularidades desse grupo de forma sucinta. Os agentes e suas respectivas funções foram alocados do seguinte modo:

- **Agente de coleta (ac0001):** Coleta uma lista de usuários que estão conectados ao sistema;
- **Agente de armazenamento (aw3001):** Informa a lista de usuários válidos do sistema;
- **Agente de análise (aa2001):** Verifica se existe algum usuário inválido nas coletas recebidas;
- **Agente de reação (ar4001):** Expulsa o usuário e bloqueia a conta para não haver mais tentativas de invasão;
- **Agente de coordenação (ao1001:** Solicitação das atividades de coleta e de reação aos agentes.

A Tabela 5.12 mostra o trecho principal do código fonte desenvolvido para o agente de coleta. A principal funcionalidade que é a própria coleta de usuários conectados é mostrada em destaque.

Tabela 5.12 - Trecho do código fonte do agente de coleta (ConnectedUserCollector.java)

```
public class ConnectedUserCollector extends BasicCollector {


    protected ContentElement collect(Agent agent, AID analyst){

        List users = new ArrayList();
        User currUser;
        String user;

        String cmd[] = {"/bin/bash","-c","/bin/ps aux | /bin/awk
'$1!=\"USER\" {print $1}' | /bin/sort | /usr/bin/uniq"};

        Iterator it = Utils.runSystemCommand(cmd).iterator();

        while ( it.hasNext() ){

            currUser = new User();
            currUser.setName(it.next().toString());
            users.add(currUser);
         }

        Collect collect = new Collect();
        collect.setDate(20050601); //acox
        collect.setLocation(agent.here().getName());

        CollectConnectedUsers ccu = new CollectConnectedUsers();
        ccu.setUsers(users);
        ccu.setCollect(collect);
        return ccu;

    }
                                    .
                                    .
}
```

Para fazer essa coleta foi utilizado o componente de execução de comandos nativos (`Utils.runSystemCommand`), aonde foi passado como parâmetro a seguinte construção Unix:

```
/bin/ps aux|/bin/awk '$1!=\"USER\" {print $1}'|/bin/sort |/usr/bin/uniq
```

Esse comando lista todos os usuários conectados em ordem alfabética, baseando-se nos processos de sistema em execução. Dessa forma, não importa a qual serviço o usuário está conectado. Por exemplo, um usuário que usasse os serviços de `ftp`, `ssh` ou, até mesmo, que ingressasse diretamente no sistema, seria identificado.

Em relação ao agente de armazenamento, apenas foi criado e retornado um `ContentElement` contendo uma lista de usuários válidos definida pela classe `KnowledgeValidUsers`, como consta no código fonte mostrado na Tabela 5.13.

Tabela 5.13 - Trecho do código fonte do agente de armazenamento (ValidUserWarehouse.java)

```
public class ValidUserWarehouse extends BasicWarehouse {
                                        .
                                        .
    protected ContentElement query(){

        List users = new ArrayList();
        User user = new User();
        user.setName("dbus"); users.add(user);
        user.setName("luiz"); users.add(user);
        user.setName("postfix"); users.add(user);
        user.setName("root"); users.add(user);
        user.setName("rpc"); users.add(user);
        user.setName("sshd"); users.add(user);
        user.setName("xfs"); users.add(user);

        Knowledge knowledge = new Knowledge();
        knowledge.setDate(2);

        KnowledgeValidUsers kvu = new KnowledgeValidUsers();
        kvu.setUsers(users);
        kvu.setKnowledge(knowledge);

        return kvu;

    }
                                        .
                                        .

}
```

Já no agente de análise, foi implementado um algoritmo que compara a lista de usuários válidos (`KnowledgeValidUsers`) com a lista de usuários conectados (`CollectConnectedUsers`), como mostrado em destaque no código fonte presente na Tabela 5.14. Um alarme foi emitido sempre que essas duas listagens não fossem idênticas, significando que existia um usuário não autorizado no sistema.

Tabela 5.14 - Trecho do código fonte do agente de análise (AnomalousUserAnalyst.java)

```
public class AnomalousUserAnalyst extends BasicAnalyst {
                                    .
    protected ContentElement analyse(ContentElement collect,
ContentElement knowledge){

        CollectConnectedUsers ccu = (CollectConnectedUsers)collect;
        KnowledgeValidUsers kvu = (KnowledgeValidUsers)knowledge;
        List listDiffUsers = new ArrayList();
        List listCollect = ccu.getUsers();
        List listKnowledge = kvu.getUsers();
        boolean userIsValid = false;

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

            userIsValid = false;

            for (int j = 0 ; j < listKnowledge.size() ; j++ ){

               String colName =((User)listCollect.get(i)).getName();
               String knoName =((User)listKnowledge.get(j)).getName();

               if (colName.compareTo(knoName)==0)userIsValid = true;

            }

            if (!userIsValid) listDiffUsers.add(listCollect.get(i));
        }

        if (listDiffUsers.size() != 0){

            Iterator it = listDiffUsers.iterator();

            while ( it.hasNext() ){
                User user = (User)it.next();
                String name = user.getName();
                int uid = user.getUid();
            }

            Alarm al = new Alarm();
            al.setDate(20);
            al.setDescription("Usuário anômalo encontrado!");
            al.setLocation(ccu.getCollect().getLocation());
            al.setLevel(1);
            AlarmAnomalousUsers aau = new AlarmAnomalousUsers();
            aau.setUsers(listDiffUsers);
            aau.setAlarm(al);

            return aau;
        } else
            return null;
    }
                                    .
                                    .
}
```

Por fim, o agente de reação foi implementado fazendo uso também do componente de execução de comandos nativos, como mostrado em destaque na Tabela 5.15. Esse comando Unix, faz com os usuários especificados sejam banidos do sistema por meio do encerramento dos seus processos.

Tabela 5.15 - Trecho do código fonte do agente de reação (BanishUserReactor.java)

```
public class BanishUserReactor extends BasicReactor {

    protected void react(Concept action){

        if (action instanceof RequestReactionBanishUser ){

            RequestReactionBanishUser rrbu =
(RequestReactionBanishUser) action;

            Iterator itUsers = rrbu.getAllUsers();

            while (itUsers.hasNext()){

                  String cmd[] = {"/bin/bash","-c","ps aux | awk ' {
if ($1 == \"" + ((User)itUsers.next()).getName() + "\") system(\"kill
-9 \" $2)}'"};

                  Utils.runSystemCommand(cmd);

                  Utils.printMessage(this.getAID(),null,new
String[]{"banish_user"});

            }
        }
    }
                                    .
                                    .
}
```

5.3 - PROCEDIMENTOS DE VALIDAÇÃO DOS CENÁRIOS

5.3.1. Configuração do ambiente

Para o preparo do ambiente de validação proposto na seção 5.1, foram configurados três computadores do tipo PC sobre um barramento físico Ethernet de 100 Mbps utilizando o protocolo de comunicação TCP/IP. Sobre essa estrutura foram criados os containeres de agentes em cada servidor, seguindo o esquema mostrado na Figura 5.13.

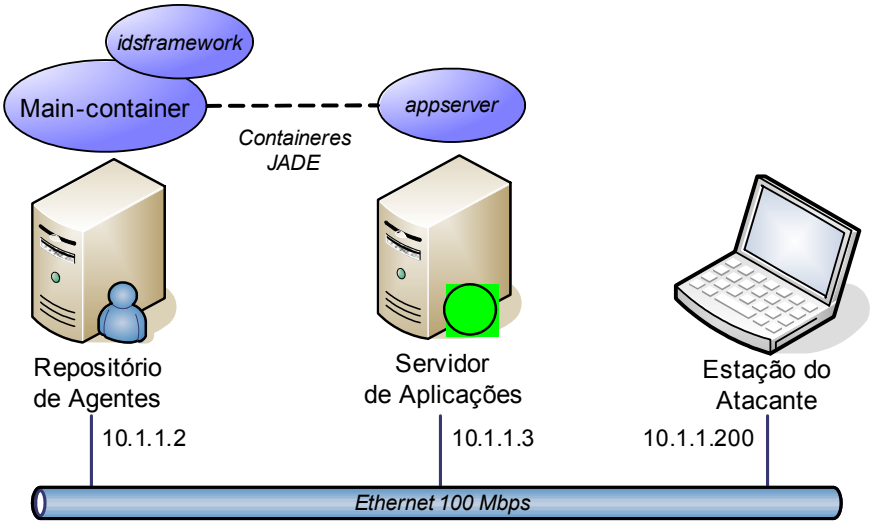


Figura 5.13 - Esquema de criação dos containeres

A Tabela 5.16 mostra os detalhes de configuração de cada computador, incluindo os serviços e ferramentas que foram instalados para a execução dos grupos de agentes e para a simulação dos ataques.

Tabela 5.16 - Configurações dos computadores do ambiente

Computador	**Configurações**	
Repositório de agentes	***Hardware***	Intel Celeron 1.30 GHz 504 MB de RAM Intel(R) PRO/100 VE Network Connection.
	Rede	`hubble` (10.1.1.2 / 8)
	Sistema	Microsoft Windows XP Professional, Service Pack 2
	Serviços	Java 2 Runtime Environment, SE (build 1.5.0-b64) [67] JADE 3.3 [68] Idsframework 0.1 (idsframework.jar)
	Container	`Main-container` `idsframework`
Servidor de aplicações	***Hardware***	Intel Pentium III 750 MHz 200 MB de RAM Realtek 8139 1/100
	Rede	`sputnik` (10.1.1.3 / 8)
	Sistema	Fedora Core 3, Linux 2.6.9-1-667 i386
	Serviços	Apache 2.0.52 [63] OpenSSH 3.9p1 [64] ProFTP 1.2.10 [62] Java 2 Runtime Environment, SE (build 1.5.0_02-b09) [67] JADE 3.3 [68]
	Container	`appserver`
Estação do atacante	***Hardware***	Intel Celeron 1.30 GHz 504 MB de RAM Intel(R) PRO/100 VE Network Connection.

	Rede	`anik` (10.1.1.2 / 8)
	Sistema	Microsoft Windows XP Professional, Service Pack 2
	Serviços	Java 2 Runtime Environment, SE (build 1.5.0-b64) [67]
	Ferramentas	Nmap 4.01 [61] PuTTY 0.55 [65] WinSCP 3.7.1 (build 257) [66]

No Servidor de Aplicações e no Repositório de Agentes foram configurados basicamente o ambiente de execução Java, o JADE e as bibliotecas do *framework* da seguinte maneira:

a) Instalação do ambiente de execução J2RE 1.5 [67] e inclusão do diretório dos executáveis na variável de ambiente `$PATH` do sistema operacional;
b) Instalação do JADE 1.3 [68] e ajuste da variável de ambiente `$CLASSPATH` com a adição dos seguintes pacotes: `Base64.jar`, `iiop.jar`, `jade.jar`, `jadeTools.jar`, `dnsns.jar` e `http.jar`;
c) Inclusão das bibliotecas do *framework* na variável de ambiente `$CLASSPATH`: `activation.jar`, `idsframework.jar`, `joone-engine.jar`, `mail.jar` e `net.sourceforge.jpcap-0.01.16.jar`.

Adicionalmente, no Servidor de Aplicações foram instalados alguns serviços para a simulação do acesso normal dos usuários. Foi feita a instalação do servidor *web* Apache 2.0.52 [63], do servidor de FTP ProFTP 1.2.10 [62] e do servidor de terminal remoto seguro OpenSSH 3.9p1 [64], ambos seguindo os roteiros de uma instalação padrão presentes nos respectivos manuais.

No Repositório de Agentes foi criado o container principal (Main-Container), responsável pela hospedagem dos agentes de controle do próprio JADE: o AMS (Agent Management System), o DF (Directory Facilitator) e o RMA (Remote Agent Management GUI) que é uma interface gráfica de administração. Dessa forma, foi executado o seguinte comando para o carregamento do Main-Container via terminal:

```
> java jade.Boot -gui
```

Com isso, o Main-Container foi criado e o RMA foi exibido conforme mostrado na Figura 5.14.

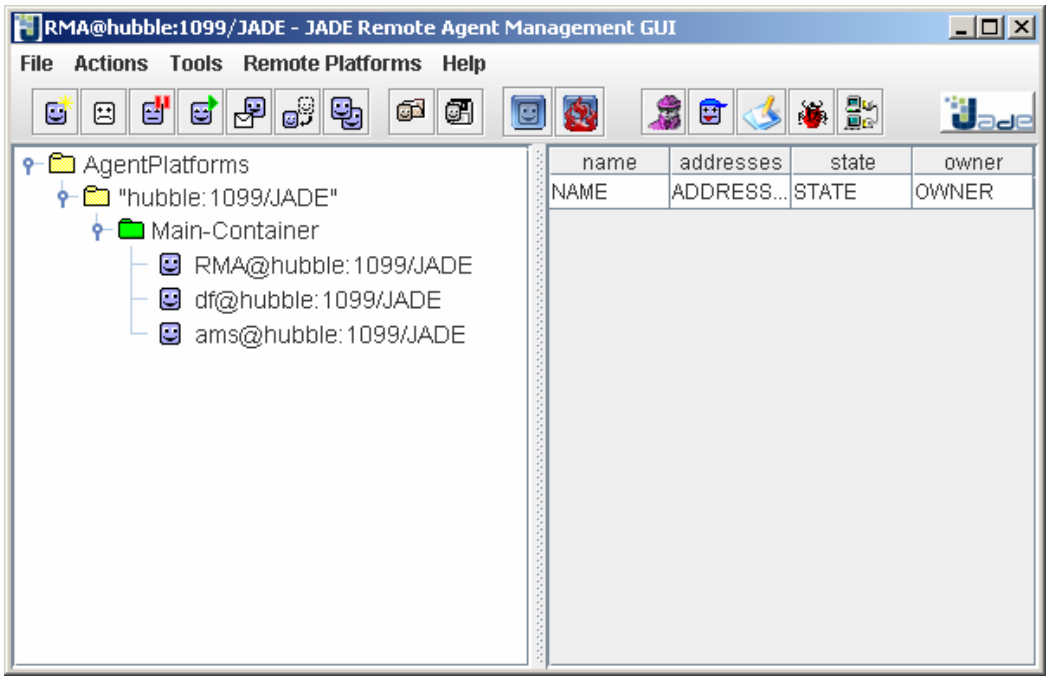


Figura 5.14 - Criação do Main-Container com o RMA, DF e AMS

A Tabela 5.17 mostra o preâmbulo de inicialização que foi obtido na criação do *Main-container*.

Tabela 5.17 – Registro de inicialização dos Main-container

```
22/04/2006 09:14:34 jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 3.3 - 2005/03/02 16:11:05
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.cselt.it/
----------------------------------------
22/04/2006 09:14:35 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
22/04/2006 09:14:35 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
22/04/2006 09:14:36 jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://hubble:7778/acc
22/04/2006 09:14:36 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
22/04/2006 09:14:36 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
22/04/2006 09:14:36 jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@JADE-IMTP://hubble is ready.
------------------------------------------------
```

O próximo passo foi a criação do container do Servidor de Aplicações, denominado como `appserver`. Para isso, foi executado o seguinte comando via terminal:

```
> java jade.Boot –container –host 10.1.1.2 –container-name appserver
```

Em relação ao comando executado, o parâmetro `container` indica que nesse local deve ser criado um novo container, e não um Main-container que é a opção padrão. O parâmetro `host` indica o endereço IP da máquina que hospeda o Main-container, que nesse caso foi o 10.1.1.2. O parâmetro `container-name` indica qual deve ser o nome desse novo container, que nesse caso foi o nome `appserver`.

O último container criado foi o específico para os agentes do *framework* chamado de `idsframework`. Para isso foi executado o seguinte comando, também via terminal:

```
> java br.unb.idsframework.Run
```

Além de essa chamada criar o container `idsframework`, também inicializa os grupos de agentes definidos na configuração do *framework*. Sendo assim, esse processo retornou uma saída com as primeiras mensagens dos agentes, conforme iam sendo criados, como mostrado na Tabela 5.18. Nesse ponto, pôde ser observado o procedimento inicial de cadastro dos agentes no diretório facilitador.

Tabela 5.18 – Registro de inicialização dos agentes do *framework*

```
22/04/2006 20:46:58 jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 3.3 - 2005/03/02 16:11:05
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.cselt.it/
----------------------------------------
22/04/2006 20:46:58 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
22/04/2006 20:46:58 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
22/04/2006 20:46:59 jade.mtp.http.HTTPServer <init>
WARNING: Port 7778 is already in used, selected another one
22/04/2006 20:46:59 jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://hubble:1534/acc
22/04/2006 20:46:59 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
22/04/2006 20:46:59 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
```

```
22/04/2006 20:46:59 jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container idsframework@JADE-IMTP://hubble is ready.
------------------------------------------------
idsframework: [aw3002] estß ativo  :::: Sat Apr 22 20:46:59 BRT 2006
idsframework: [ac0002] estß ativo  :::: Sat Apr 22 20:46:59 BRT 2006
idsframework: [aa2002] estß ativo  :::: Sat Apr 22 20:46:59 BRT 2006
idsframework: [ar4002] estß ativo  :::: Sat Apr 22 20:46:59 BRT 2006
idsframework: [ao1001] estß ativo  :::: Sat Apr 22 20:46:59 BRT 2006
idsframework: [ar4002] se cadastrou no DF  :::: Sat Apr 22 20:47:00 BRT 2006
idsframework: [aw3002] se cadastrou no DF  :::: Sat Apr 22 20:47:00 BRT 2006
idsframework: [ao1001] se cadastrou no DF  :::: Sat Apr 22 20:47:00 BRT 2006
idsframework: [aa2002] se cadastrou no DF  :::: Sat Apr 22 20:47:00 BRT 2006
idsframework: [ac0002] se cadastrou no DF  :::: Sat Apr 22 20:47:01 BRT 2006
```

Após essas configurações, o ambiente ficou preparado para o início dos testes. O RMA foi atualizado com os outros containeres conforme apresentado na Figura 5.15, aonde pôde ser observada no container `idsframework` a presença de todos os agentes implementados.

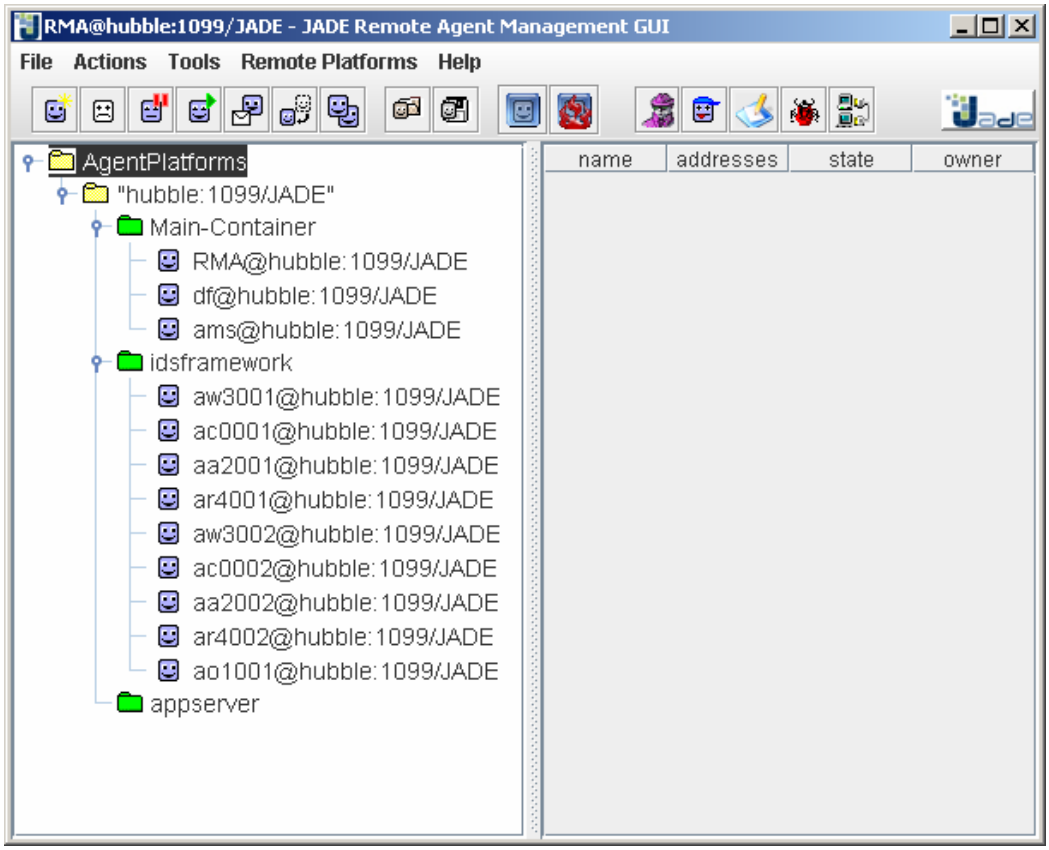


Figura 5.15 - Visualização de todos containeres no RMA

As próximas seções relatarão os procedimentos de testes de cada cenário de detecção em particular juntamente com os resultados obtidos. Nesse momento ainda não será feita uma análise desses resultados, sendo que esse tópico ficará a cargo da última seção desse capítulo.

5.3.2. Testes do cenário de detecção de usuários anômalos

Para a execução dos testes foi utilizada a ferramenta de depuração de mensagens do próprio JADE, que também é um agente chamado *sniffer*. O objetivo do primeiro teste foi avaliar o preâmbulo de organização dos agentes desse grupo e realizar uma simulação de invasão por um usuário desconhecido.

A Figura 5.16 mostra o conjunto de mensagens iniciais para a organização do grupo de agentes.

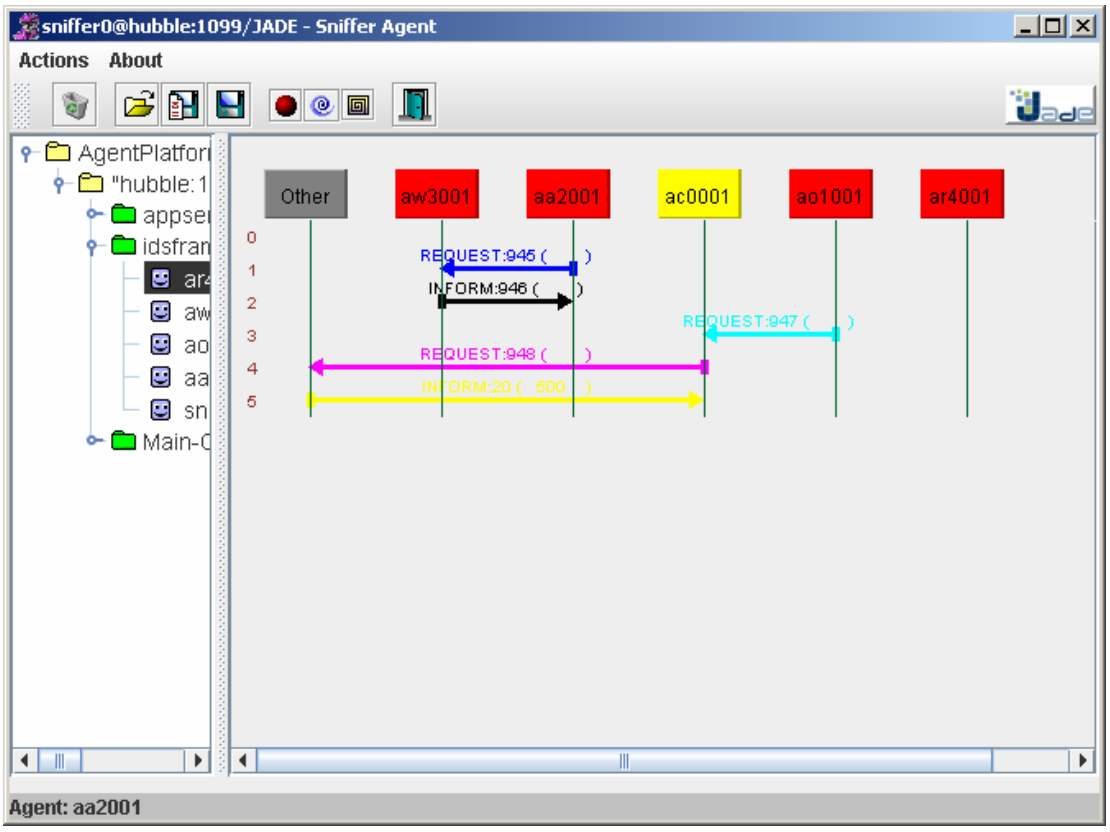


Figura 5.16 - Monitoramento das mensagens iniciais dos agentes

A primeira mensagem (REQUEST:945) foi a solicitação de conhecimento do agente de análise (aa2001) para o agente de armazenamento (aw3001). Como resposta o agente de armazenamento enviou uma mensagem com uma lista de usuários válidos (INFORM:946). A Tabela 5.19 demonstra o conteúdo dessas mensagens ACL.

Tabela 5.19 – Conteúdos das mensagens do processo de obtenção de conhecimento

Solicitação de conhecimento (REQUEST:945)

```
((action
     (agent-identifier
       :name aw3001@hubble:1099/JADE)
     (RequestKnowledgeValidUsers)))
```

Conhecimento enviado (INFORM:946)

```
((KnowledgeValidUsers
     (sequence
       (User :name apache :uid 0)
       (User :name dbus :uid 0)
       (User :name luiz :uid 0)
       (User :name mysql :uid 0)
       (User :name postfix :uid 0)
       (User :name root :uid 0)
       (User :name rpc :uid 0)
       (User :name sshd :uid 0)
       (User :name xfs :uid 0))
    (Knowledge :date 2006-04-22)))
```

Após isso o agente de coordenação fez a solicitação de coleta (REQUEST:947) ao agente de coleta informando os parâmetros da coleta como o local, agendamento de tempo e quantidade de amostras por coleta. O conteúdo dessa mensagem pode ser observado na Tabela 5.20.

Tabela 5.20 – Conteúdo da mensagem de solicitação de coleta ao agente de coleta

Solicitação de coleta (REQUEST:947)

```
((action
     (agent-identifier
       :name ac0001@hubble:1099/JADE)
     (RequestCollect
       :location appserver
       :finishtime 0
       :delay 1000
       :analyst
         (agent-identifier
           :name aa2001@hubble:1099/JADE)
       :duration 0
       :starttime 0
       :quantity 1000)))
```

Uma vez que o agente de coleta recebeu a solicitação do agente de coordenação, esse teve de se moveu para o container especificado, que no caso foi o `appserver`, e iniciou a coleta dos usuários conectados. Nesse meio tempo, foi feita a simulação de invasão por meio do acesso remoto de um usuário denominado como `invasor`, conforme a Figura 5.17.

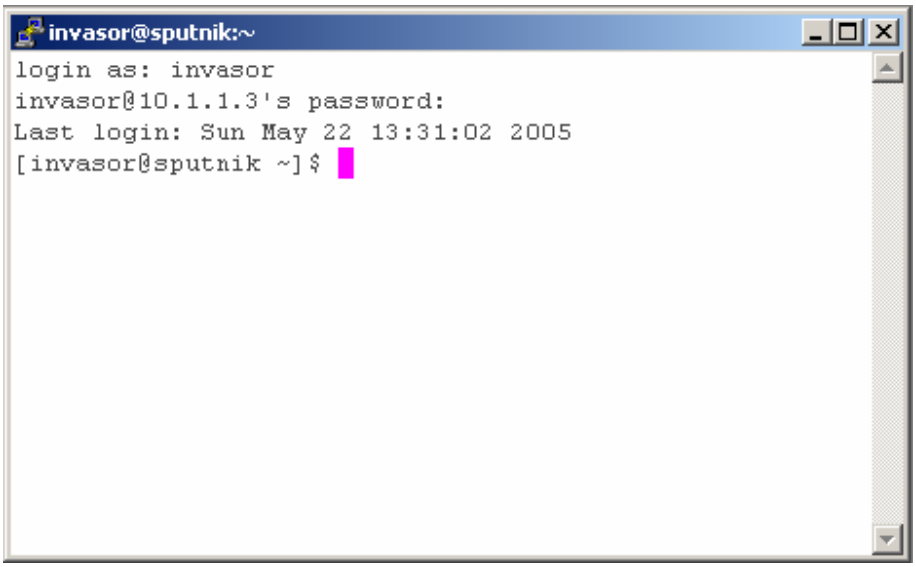


Figura 5.17 - Terminal simulando o acesso de um invasor

A Figura 5.18, retirada do agente de *sniffer*, mostra as diversas mensagens enviadas entre os agentes em uma situação normal e uma situação de ataque proveniente do acesso remoto do invasor.

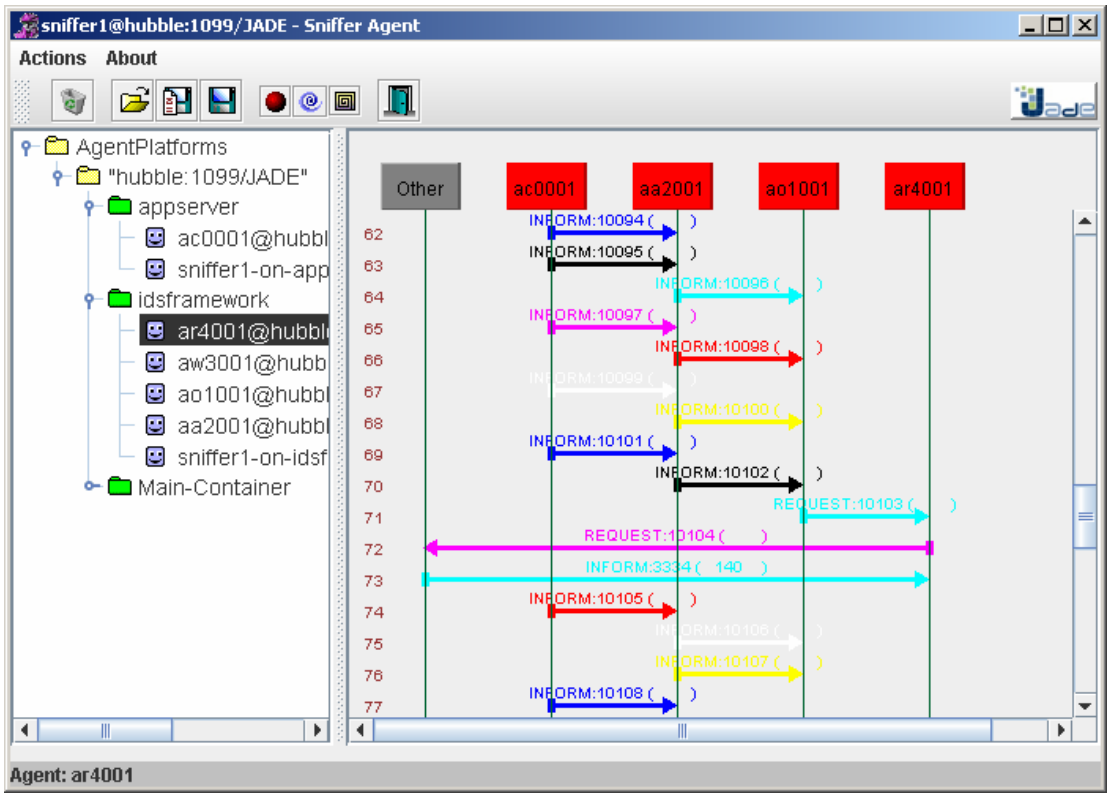


Figura 5.18 - Nova série de mensagens de coleta de dados

A primeira mensagem de coleta (INFORM:1094), enviada do agente de coleta (ac0001) para o agente de análise (aa2001), se referia a uma mensagem normal contendo os usuários válidos conectados ao servidor com o conteúdo mostrado na Tabela 5.21.

Tabela 5.21 - Conteúdo da mensagem de coleta de usuários conectados

Coleta com usuários válidos (INFORM:1094)

```
((CollectConnectedUsers
     (sequence
       (User :name apache :uid 0)
       (User :name dbus :uid 0)
       (User :name root :uid 0)
       (User :name rpc :uid 0)
       (User :name sshd :uid 0)
       (User :name xfs :uid 0))
     (Collect :location appserver :date 20050601)))
```

Em contrapartida, a mensagem de coleta seguinte (INFORM:10095) indicou uma situação de invasão. No conteúdo dessa mensagem foi encontrado um usuário anômalo como mostrado em destaque na Tabela 5.22.

Tabela 5.22 - Conteúdo da mensagem de coleta com um usuário anômalo

Coleta com um usuário anômalo (INFORM:10095)

```
((CollectConnectedUsers
    (sequence
      (User :name apache :uid 0)
      (User :name dbus :uid 0)
      (User :name invasor :uid 0)
      (User :name luiz :uid 0)
      (User :name mysql :uid 0)
      (User :name postfix :uid 0)
      (User :name root :uid 0)
      (User :name rpc :uid 0)
      (User :name xfs :uid 0))
    (Collect
      :location appserver
      :date 20050601)))
```

Foi iniciado então o processo de reação do grupo. A mensagem de coleta foi avaliada pelo agente de análise (aa2001) e um alarme (INFORM:10096) foi enviado ao agente de coordenação (ao1001) que informou a presença do usuário `invasor`, conforme mostrado em destaque na Tabela 5.23..

Tabela 5.23 - Conteúdo da mensagem de alarme de usuário anômalo

Alarme de presença de usuário anômalo (INFORM:10096)

```
((AlarmAnomalousUsers
     (sequence
       (User :name invasor :uid 0))
     (Alarm
       :location appserver
       :date 2006-04-22
       :description "Usuarios anomalos foram detectados ! "
       :level 1.0)))
```

Após receber o alarme, o agente de coordenação solicitou a confirmação das atividades de reação por meio da caixa de diálogo mostrada na Figura 5.19. Esse procedimento foi colocado apenas para se ter visibilidade das ações do agente, porém esse processo poderia ser automático.

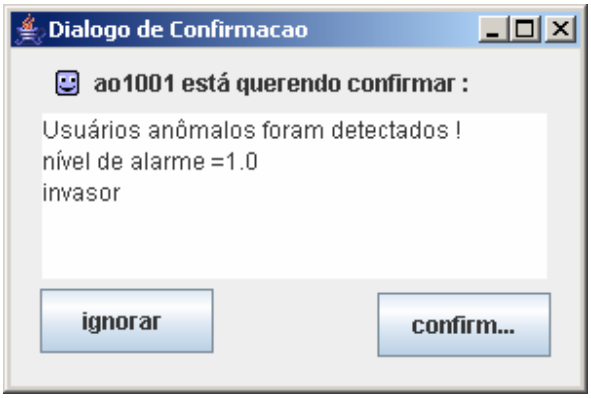


Figura 5.19 - Diálogo de confirmação de reação

Após a confirmação da reação, o agente de coordenação enviou uma solicitação de reação ao agente de reação (ar4001) por meio da mensagem (REQUEST:10103) cujo conteúdo está mostrado na Tabela 5.24.

Tabela 5.24 - Conteúdo da mensagem de solicitação de reação

Solicitação de reação (REQUEST:10103)

```
((action
 (agent-identifier
   :name ar4001@hubble:1099/JADE)
 (RequestReactionBanishUser
   :location appserver
   :users
     (sequence
       (User :name invasor :uid 0)))))
```

Por fim, o agente de reação baniu o invasor do sistema e bloqueou a conta para não permitir novos acessos. A Figura 5.20 mostra o terminal remoto sendo encerrado após a ação desse agente.

Figura 5.20 - Terminal remoto do invasor sendo encerrado

5.3.3. Testes do cenário de detecção de *port scanning*

A Figura 5.21 mostra o conjunto de mensagens iniciais para a organização do grupo de agentes de detecção de *port scanning*.

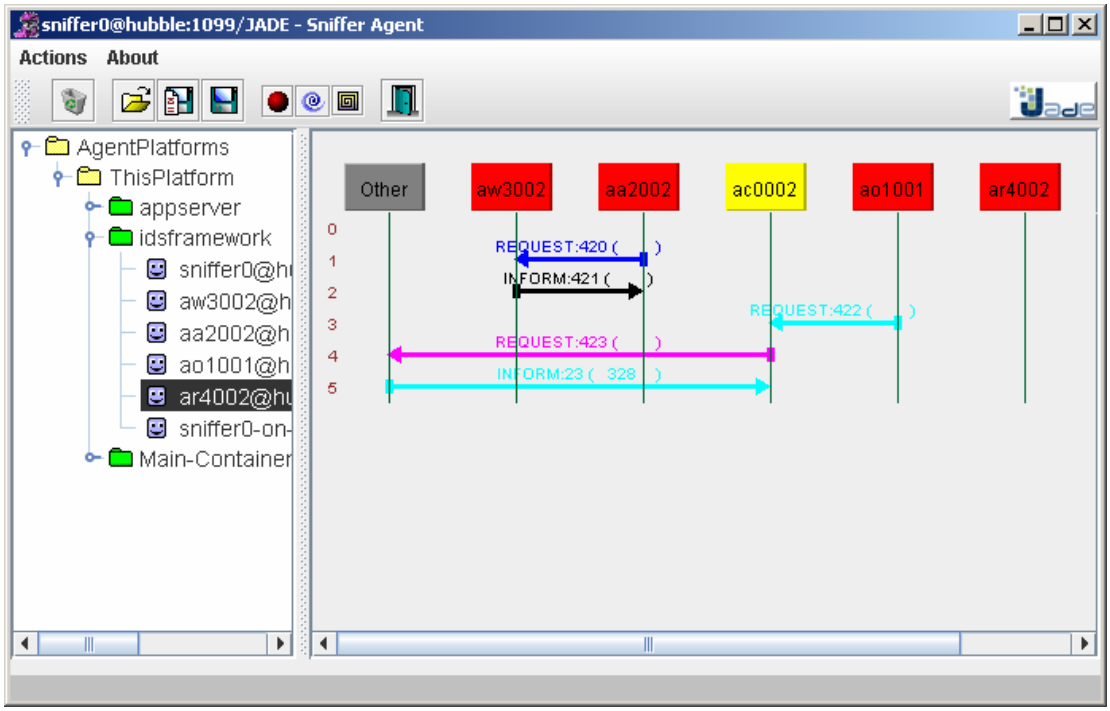


Figura 5.21 - Monitoramento das mensagens iniciais dos agentes

A primeira mensagem (REQUEST:420) foi a solicitação de conhecimento do agente de análise (aa2002) para o agente de armazenamento (aw3002). Como resposta o agente de armazenamento enviou uma mensagem com uma lista de padrões de pacotes de *port scanning* (INFORM:421). A Tabela 5.25 mostra o conteúdo dessas mensagens ACL.

Tabela 5.25 - Conteúdos das mensagens de atualização de conhecimento

Solicitação de conhecimento (REQUEST:420)

```
((action
     (agent-identifier :name aw3002@hubble:1099/JADE)
                 (RequestKnowledgeAnomalousPackets)))
```

Conhecimento enviado (INFORM:421)

```
((KnowledgeAnomalousPackets
     (sequence 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 -34.17662811279297 5.194009780883789 -
14.518896102905273 12.562366485595703 -3.4134910106658936)
(Knowledge :date 2)
(sequence
47.585166931152344 -13.347997665405273 -6.391445159912109
-5.8088297843933105 -23.73871421813965 -5.134727954864502
                                  .
                                  .
0.47210800647735596 0.31481799483299255 0.038777999579906464
-45.906944274902344 25.155376434326172 8.871492385864258
10.737855911254883)))
```

Após isso o agente de coordenação fez a solicitação de coleta (REQUEST:422) ao agente de coleta (ac0002) informando os parâmetros da coleta. O conteúdo dessa mensagem pode ser observado na Tabela 5.26.

Tabela 5.26 - Conteúdo da mensagem de solicitação de coleta

Solicitação de coleta (REQUEST:422)

```
((action
     (agent-identifier
       :name ac0002@hubble:1099/JADE)
     (RequestCollect
       :location appserver
       :finishtime 0
       :delay 2000
       :analyst
         (agent-identifier
           :name aa2002@hubble:1099/JADE)
       :duration 0
       :starttime 0
       :quantity 1000)))
```

Do mesmo modo, o agente de coleta se moveu para o container `appserver`, e iniciou a coleta dos pacotes de rede. Para realizar os testes necessários, foram gerados cinco modalidades de tráfego para esse servidor: tráfego de acesso HTTP, tráfego de FTP, tráfego de terminal remoto SSH, tráfego misto e o tráfego gerado pelo *port scanning*.

O tráfego de HTTP foi gerado por meio do acesso aleatório às páginas de uma aplicação *web* (http://10.1.1.3/phpmyadmin) de gerenciamento de banco de dados. A aplicação acessada é mostrada na Figura 5.22.

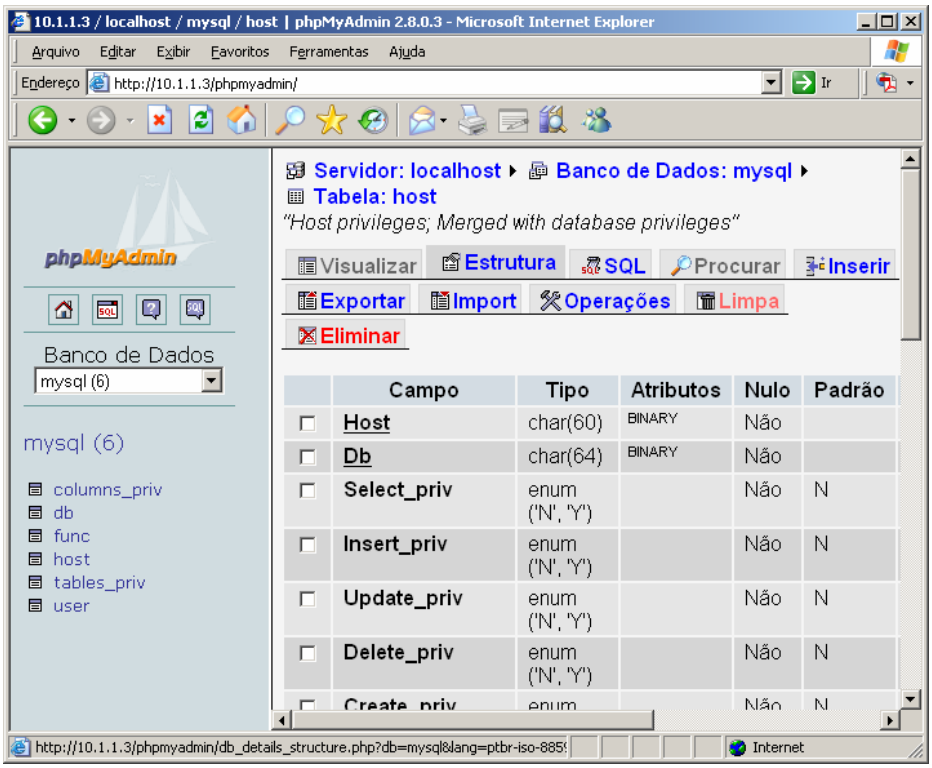


Figura 5.22 - Aplicação web que gerou o tráfego HTTP

O tráfego de FTP foi gerado por uma cópia remota de um diretório inteiro do servidor de aplicações (ftp://10.1.1.3/mestrado). A Figura 5.23 mostra o processo de cópia executado.

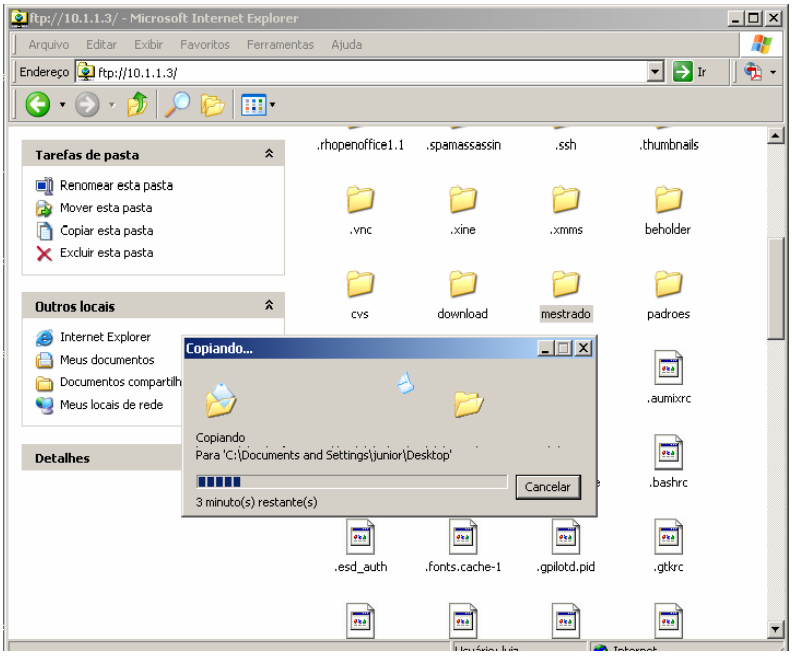


Figura 5.23 - Cópia de arquivos que gerou o tráfego FTP

O tráfego SSH do terminal remoto foi gerado com o comando `locate` procurando por palavras freqüentes como `zip` que gerou bastante texto na saída do terminal. A Figura 5.24 mostra o terminal remoto executando uma das buscas.

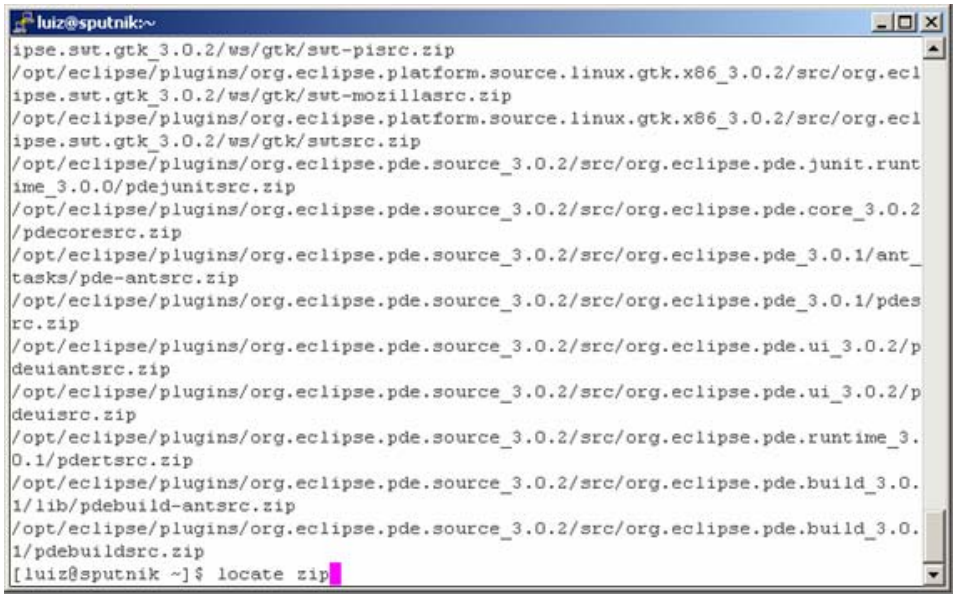


Figura 5.24 – Terminal remoto que gerou o tráfego SSH

O tráfego misto foi gerado com a execução desses três procedimentos simultaneamente durante 1 minuto.

O tráfego de *port scanning* foi gerado com as opções padrões do `nmap` da seguinte forma:

```
> nmap 10.1.1.3
```

Os resultados obtidos em cada tipo de tráfego são relatados na Tabela 5.27. Em nenhuma das situações normais houve a geração de falsos positivos. Já na situação de *port scanning* os agentes agiram como era esperado, já que o nível de alarme superou o limiar estabelecido de 0,95.

Tabela 5.27 - Resultados obtidos nas gerações de tráfego

Tráfego	Nº Pacotes Analisados	Duração	Nível de Alarme Médio
HTTP	24.000	2 min	0,62
FTP	137.000	3 min	0,38
SSH	5.000	20 s	0,56
Misto	52.000	1 min	0,49
Port Scanning	4.000	15 s	0,99

No caso do *port scanning*, o agente de reação enviou um alarme por e-mail para o administrador como mostrado na Figura 5.25.

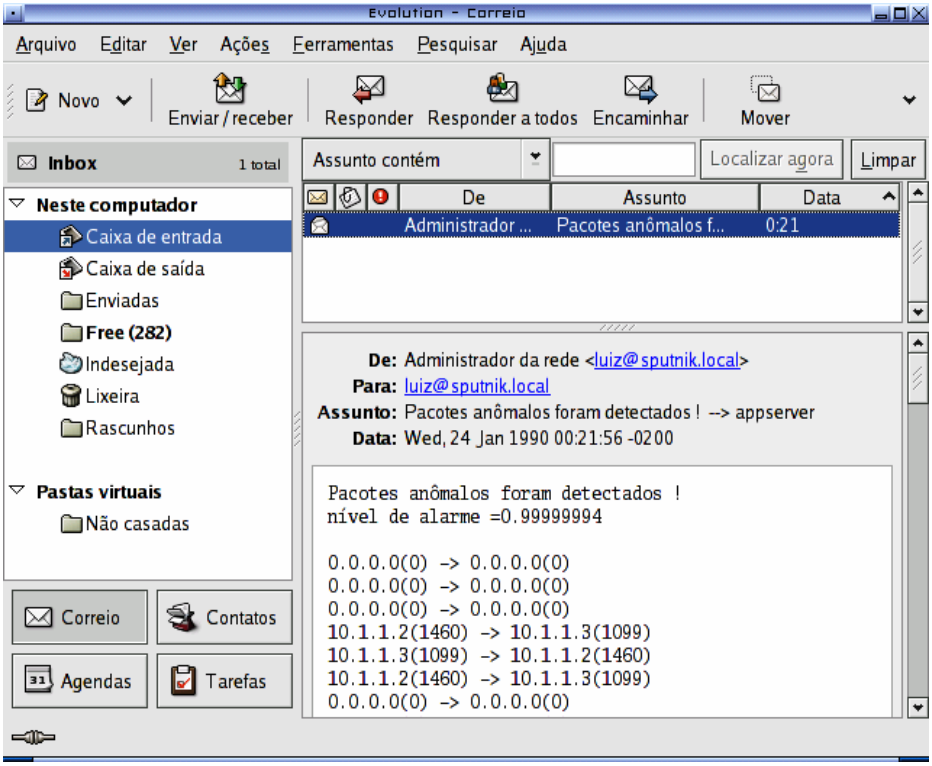


Figura 5.25 - E-mail enviado pelo agente de reação

5.4 - AVALIAÇÃO DOS RESULTADOS

Como pode ser observado no primeiro cenário, o grupo de detecção de usuários anômalos agiu conforme o esperado. Inicialmente o agente de coleta iniciou o envio de pacotes de dados contendo os usuários para o agente de análise que já dispunha do conhecimento fornecido pelo agente de armazenamento. Ao ser detectado um usuário anômalo, o agente de análise gerou um alarme para o agente de coordenação que providenciou as contramedidas por meio do agente de reação. Rapidamente o usuário anômalo foi expulso do sistema.

Da mesma forma, ao serem detectados os pacotes contendo padrões de *port scanning*, o grupo de agentes se articulou no sentido de gerar um alarme via e-mail para o administrador.

6 - CONCLUSÕES E RECOMENDAÇÕES

A fim de atender às motivações desse trabalho, foi desenvolvido, em nível de protótipo, um *framework* para construção de sistemas de detecção de intrusão, fazendo uso de uma arquitetura baseada em grupo de agentes de software especializados. Para isso, foram necessárias as etapas de concepção, implementação e validação do *framework*, etapas clássicas da Engenharia de Software.

Para a concepção, foi necessário estabelecer as características desejáveis do *framework* e eleger uma arquitetura de software. As características desejáveis visaram beneficiar tanto o processo construtivo, fornecendo ferramentas e metodologias aos desenvolvedores para acelerar o desenvolvimento, quanto o próprio sistema de detecção de intrusão construído sobre o *framework*, por meio de uma arquitetura flexível, escalável e robusta. Essa arquitetura foi estabelecida após a pesquisa de outras propostas que levaram ao desenvolvimento baseado em grupos formados por agentes especializados (agente de coleta, agente de análise, agente de armazenamento, agente de reação e agente de coordenação).

A implementação fez uso de um modelo orientado a objetos e orientado a agentes de software, por meio de tecnologias baseadas em Java. Dessa forma, esse modelo garantiu o nível de reusabilidade necessário para caracterizar a solução como um *framework* extensível. Além disso, o uso de uma plataforma de agentes permitiu que fossem aproveitadas as habilidades de comunicação, mobilidade e autonomia inerentes dos agentes, fornecendo um modelo mais natural para a programação das atividades paralelas e distribuídas requeridas pelos sistemas de detecção de intrusão.

A validação prática dos cenários pôde comprovar a viabilidade de se construir facilmente os grupos de agentes especializados usando as ferramentas e metodologias do *framework*. Também comprovou sua flexibilidade com o uso de diferentes técnicas auxiliares, como a aplicação de redes neurais artificiais para a detecção e a operação do tipo *network-based* e *host-based*.

Dentre as colaborações mais importantes desse trabalho, podem ser citadas:

- Exploração de uma arquitetura baseada em grupos de agentes especializados em cada atividade do SDI;
- Implementação de um protótipo funcional que pode ser aproveitado em próximos trabalhos;
- Proposta de um roteiro de modelagem de ontologias para a formulação da comunicação entre os agentes usando ferramentas de apoio;
- Aplicação prática do *framework* em diferentes cenários para a compreensão e auxílio em novas implementações;
- Apresentação das fases necessárias para o uso de redes neurais artificiais (tratamento da massa de dados, concepção da topologia, treinamento e teste);
- Introdução de metodologias recentes de apoio à modelagem de sistemas baseados em agentes, como a AUML;

Como trabalhos futuros para essa pesquisa, são sugeridos os seguintes tópicos:

- Aumento da capacidade de gerenciamento do agente de coordenação, também utilizando as técnicas da Inteligência Artificial. Dessa forma, esse agente poderia atuar de forma mais estratégica olhando a situação global da rede e prevenindo ataques distribuídos e coordenados;
- Desenvolver mecanismos de aprendizagem em tempo real que evoluíssem os agentes de armazenamento para verdadeiros agentes de conhecimento;
- Maior aproximação da abordagem MAS, por meio da designação de agentes mais autônomos e colaborativos;
- Estabelecer interfaces para o desacoplamento dos componentes de serviços. Com isso, eles poderiam ser desenvolvidos, instalados e removidos separadamente;
- Levantamento e implementação de mais componentes de serviços necessários para os sistemas de detecção de intrusão;

- Melhoria na segurança do próprio *framework* (resistência a subversão). Uma dessas iniciativas seria a proteção da comunicação que ocorre em claro por meio de protocolos como o SSL (Secure Socket Layer). Também a implementação de um processo de autenticação dos agentes para evitar o ingresso de agentes maliciosos;
- Em relação ao grupo de detecção de *port scanning*, poderia ser implementadas várias redes neurais artificiais em paralelo, uma para cada tipo de port scanning. Isso poderia melhorar os índices de falso positivos e falso negativos;
- Outro ponto preocupante seria a redução do número de chamadas as funções nativas do sistema operacional (captura de pacotes e leitura de processos) que comprometem a portabilidade do *framework*;
- Por fim, como essa proposta avaliou apenas as características metodológicas e funcionais do *framework*, são necessários alguns testes adicionais de cunho não funcional, como a escalabilidade, a tolerância a falhas e a performance.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Vandoorselaere, Y. *Prelude-IDS: The Hybrid IDS Framework*, 2005, http://www.prelude-ids.org. Acessado em novembro de 2004.

[2] Schnackenberg, D., Porras, P., Staniford-Chen, S. Stillman, M. e Wu, F. *The Common Intrusion Detection Framework Architecture*. http://www.isi.edu/gost/cidf/drafts/architecture.txt. Acessado em janeiro de 2005.

[3] Bace R. e Mell, P. *NIST Special Publication on Intrusion Detection Systems. Primer o intrusion detection systems – guidance*, Infidel, Inc., Scotts Valley, CA and National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf. Acessado em setembro de 2005.

[4] Bernardes, M. *Avaliação do uso de agentes móveis em segurança computacional*. Dissertação de mestrado, ICMC/USP. 1999.

[5] Crubézy, M., Musen, M., Noy, N., O'Connor, M., Timothy, R., Rubin, D, Tu, S. e Tudorache, T. *The Protégé Ontoly Editor and Knowledge Acquisition System*. Stanford Medical Informatics - Stanford University School of Medicine. http://protege.stanford.edu/. Acessado em fevereiro de 2005.

[6] De Holanda Ferreira, A. B. *Novo Aurélio Século XXI: O Dicionário da Língua Portuguesa*. 3ª Edição, Editora Nova Fronteira, 1999.

[7] International Organization for Standardization / International Eletrotechinical Commitee. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture*. International Standard 7498-2, 1989.

[8] Nakamura, E. e Geus, P. *Segurança de Redes em Ambientes Cooperativos*. 2ª edição, Editora Futura, 2003.

[9] Stallings, W. (2000). *Cryptography and Network Security – Principles and Practice*. Prentice Hall, Second Edition, 2000.

[10] Anderson, J. *Computer Security Threat Monitoring and Surveillance*. Fort Washignton, PA: James P. Anderson Co, 1980.

[11] Módulo Security Solutions. *Mailing Lists*. http://www.modulo.com.br. Atualizado em 12 de março de 1999.

[12] Gordon, L. A., Loeb, M. P., Lucyshyn, W. e Richardson, R. *Computer Crime and Security Survey 2005*. CSI/FBI. Computer Security Institute Publications, Tenth Annual. http://www.usdoj.gov/criminal/cybercrime/FBI2005.pdf. Acessado em março de 2005.

[13] Wazir, Burhan. Hacker Cries Foul Over FBI Snooping. http://guardian.co.uk/Internetnews/story/0,7369,578081,00.html. Acessado em dezembro de 2005.

[14] Nmap - Free Security Scanner For Network Exploration & Security Audits. *Guia de Referência do Nmap (Man Page)*. http://www.insecure.org/nmap/man/pt-br/. Acessado em abril de 2004.

[15] SecureXpert Labs Advisory SX-98.12.23-01. *Widespread DoS Vulnerability can Crash Systems or Disable Critical Services*. http://packetstorm.security.com/new-exploits/nmap-DoS-2.txt. Acessado em janeiro de 2004.

[16] Anonymous. *Maximum Security*. Sams Publishing, fourth edition, 2003.

[17] Lemos, R. *DDoS Attacks - one year later* . ZDNet News US. http://news.zdnet.co.uk/business/0,39020645,2084263,00.htm. Arquivo acessado de fevereiro de 2001.

[18] Cohen, F. A. *A Short Course on Computer Viruses*. New York. Wiley, 1994

[19] Nachenberg, C. *Virus Boletim.* Symantec Research Labs, 2003.

[20] Holbrook, P. e Reynolds, J. *Site Security Handbook (RFC 1244)*. Network Working Group, 1991. http://www.faqs.org/rfcs/rfc1244.html. Acessado em março de 2005.

[21] Wikipédia, a enciclopédia livre. Palavra-chave: "Guerra de Tróia". http://pt.wikipedia.org/wiki/Guerra_de_Tróia. Acessado em abril de 2005.

[22] Tanenbaum, Andrew S. *Redes de Computadores*. Tradução da Terceira edição. Editora Campus. 1997.

[23] Schneier, Bruce. *Applied Cryptografy*. Second Edition. John Willey & Sons, Inc.1995.

[24] R. Rivest. *The MD5 Message-Digest Algorithm (RFC 1321)*. MIT Laboratory for Computer Science and RSA Data Security, Inc. 1992. http://www.faqs.org/rfcs/rfc1321.html. Acessado em março de 2005.

[25] Prabhakar, A. Secure Hash Standard (FIPS PUB 180-1). U.S. Department of Commerce, 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm. Acessado em dezembro de 2004.

[26] Adams, Carlisle; Lloyd, Steve. *Understanding Public-Key Infrasctructure - Concepts, Standards and Deployment Considerations.* New Riders, 1999.

[27] Bellovin, S., and Cheswick, B. *Network Firewalls*. IEEE Communications Magazine, 1994.

[28] Chapman, D. Brent. Awichy, Elizabeth D. Building Internet Firewalls. O'Reilly & Associates, Inc. 1995.

[29] Rekhter, Y., Moskowitz, B., Karrenberg, D., De Groot, G. J., Lear, E. *Address Allocation for Private Internets (RFC 1918)*. Network Working Group, 1996. http://www.faqs.org/rfcs/rfc1918.html. Acessado em junho de 2005.

[30] Crosbie, M.; Spafford, E.H. *Defending a Computer System using Autonomous Agents*. Departament of Computer Sciences, Pardue University, 1995. (Relatório Técnico CSD-TR-95-022; Coast TR 95-02). http://www.cs.purdue.edu/homes/spaf/tech-reps;9522.ps. Acessado em janeiro de 2005.

[31] Mukherjee, B.; Heberlein, T.; Levitt, K. *Network Intrusion Detection*. IEEE Network, 8(3):26-41 (1994).

[32] Da Silva, T. A. *Um ambiente baseado em agentes de software para detecção e análise de ataques em redes de computadores*. Dissertação de Mestrado. Universidade de Brasília, 2004.

[33] Tavares, D. T. Avaliação de Técnicas de Captura para Sistemas Detectores de Intrusão. Dissertação de Mestrado. ICMC/USP, 2002.

[34] Denning, D., Edwards, D.,Jagannathan, R., Lunt, T. e Neumann, P. *A prototype IDES – a real time intrusion detection expert system*. Technical Report, Computer Science Laboratory, SRI. International. 1987.

[35] Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L. *Distributed Intrusion Detection System - Motivation, Architecture and An Early Prototype*. Computer Security Laboratory, Division of Computer Science, University of California, 1991.

[36] Porras e Neumann. *EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances*, 1997. Proc. 20th NIST-NCSC National Information System Security Conference.

[37] Zamboni, D., Aubramaniyan, J., Garcia-Fernandes, J. O. e Spafford, E. H. *Autonomous Agents For Intrusion Detection* Department Of Computer Sciences, Purdue University, 1995.

[38] Torsun, I.S., *Foundations of Intelligent Knowledge-based Systems*. London: Academic Press, 1995.

[39] Bradshaw, J. *An introduction to software agents*. AAAI Press/ The MIT Press, 1997.

[40] Franklin, S.; Graesser, A. *Is It An Agent, or Just a Program ? A taxonomy fot Autonomous Agents.* In: Proceedings of the Third International workshop on Agent Theories, Architectures, and Languages. Springer-Verlag, 1996. http://www.msci.memphis.edu/~franklin/AgentProg.html. Acessado em janeiro de 2006.

[41] Nwana, H. *Software Agents: An Overview.* Knowledge Engineering Review, 11(3):205-244. http://citeseer.nj.nec.com/nwana96software.html. Acessado em janeiro de 2006.

[42] Silva, L. A. M. *Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development Framework*. Monografia Final para Bacharel em Informática. Universidade de Fortaleza, 2003.

[43] FIPA Organization. *FIPA - Foundation for Intelligent Physical Agents.* http://www.fipa.org. Acessado em junho de 2004.

[44] Demazeau, Y. e Müller, J. P. *Descentralized Artificial Intelligence*. Proceedings of the First European Workshop on Modeling Autonomous Agents in a Multi-Agent World. Cambridge: North-Holland, 1990.

[45] Sichman et al., Demazeau, J. Y. Exploiting Social Reasoning to Deal with Agency Level Inconsistencies. Proc. ICMAS-95, 1995, p. 352-359.

[46] Odell, J. J., Parunak, H. V. D., Bauer, B. Representing Agent Interaction Protocols in UML. http://www.auml.org/auml/supplements/Odell-AOSE2000.pdf. Acessado em janeiro de 2005.

[47] *World Wide Web Consortium (W3C).* http://www.w3.org. Acessado em fevereiro de 2004.

[48] *Beangenerator for Protégé*. http://acklin.nl/beangenerator. Acessado em março de 2005.

[49] *Jpcap - A network packet capture library*. http://jpcap.sourceforge.net. Acessado em dezembro de 2004.

[50] *TCPDUMP public repository*. http://www.tcpdump.org/. Acessado em janeiro de 2005.

[51] *WinPcap: The Windows Packet Capture Library.* http://www.winpcap.org/. Acessado em novembro de 2004.

[52] J. Cannady. *Artificial neural networks for misuse detection*. In Proceedings of the 1998 National Information Systems Security Conference (NISSC'98), pages 443--456, October 5-8 1998. Arlington, VA. http://citeseer.ist.psu.edu/cannady98artificial.html. Visitado em setembro de 2004.

[53] *Joone - Java Object Oriented Neural Engine*. http://www.jooneworld.com. Acessado em abril de 2005.

[54] Birner, E. e Uzunian, A. *Biologia 3*. Editora HARBRA. 3ª Edição, 2006.

[55] *JADE - Java Agent DEvelopment Framework.* http://sharon.cselt.it/projects/jade Parma - Itália. Acessado em fevereiro de 2005.

[56] Kimberg, T., Coulouris, G. e Dollimore, J. *Distributed Systems – Concepts and Design*. 3ª Edição. Ed. Addison Wesley, 2001

[57] Bombonato, F. *Beholder – Sistema de detecção de intruso baseado em redes neurais*. Monografia final de graduação. Universidade Católica de Brasília. 2003.

[58] Cannady, J. *Artificial Neural Networks for Misuse Detection*. School of Computer and Information Sciences. Nova Southeastern University.

[59] Rocha, D. L. *Um ambiente honeynet para treinamento de redes neurais artificais*. Dissertação de Mestrado. Faculdade de Tecnologia, Departamento de Engenharia Elétrica. 2006.

[60] *EasyNN - Neural Network Software*. http://www.easynn.com/. Acessado em junho de 2005.

[61] *Nmap - Free Security Scanner For Network Exploration & Security Audits*. *Guia de Referência do Nmap (Man Page)*. http://www.insecure.org/nmap/man/pt-br/. Acessado em abril de 2004.

[62] *The ProFTPD Project Home*. http://www.proftpd.org. Acessado em julho de 2005.

[63] *The Apache Software Foundation*. http://www.apache.org. Acessado em juno de 2005.

[64] *OpenSSH*. http://www.openssh.com. Acessado em junho de 2005.

[65] *PuTTY: A Free Telnet/SSH Client.* http://www.chiark.greenend.org.uk/~sgtatham/putty. Acessado em maio de 2005.

[66] *WinSCP: Freeware SFTP and SCP client for Windows*. http://winscp.net/eng/index.php. Acessado em janeiro de 2005.

[67] *Java Platform, Standard Edition (Java SE).* http://java.sun.com/javase/index.jsp. Acessado em março de 2005.

[68] *JADE - Java Agent DEvelopment Framework*. http://sharon.cselt.it/projects/jade Parma - Itália. Acessado em fevereiro de 2005.

[69] Baker, Albert. *JAFMAS – A java-based agent framework for multiagent systems. Development and Implementation*. Cincinnati: Department of Eletrical & Computer Engineering and Computer Science University of Cincinnati, 1997. Tese de Doutorado.

[70] Gruber, T. *What is an ontology?,* 1996. http://www-ksl.stanford.edu/ kst/what-is-an-ontology.html. Acessado em julho de 2005.

[71] Genesereth, M. R., Nilson, L. *Logical foundation of AI*. San Francisco, Los Altos, Califórnia : Morgan Kaufman, 1987.

[72] Guarino, N. *Understanding, Building And Using Ontologies*. *In*: Proceedings Of Knowledge Acquisition For Knowledge-Based Systems Workshop. 10. 1996. http://ksi.cpsc.ucalgary.ca/kaw/kaw96/guarino/guarino.html#heading4. Acessado em setembro de 2005.

[73] Borst, W. N. *Construction of engineering ontologies*. 1997. Tese de Doutorado. University of Twente, Enschede, 1997. http://www.ub.utwente.nl/webdocs/inf/1/t0000004.pdf. Acessado em: abril de 2005.

[74] Mauricio B. A., Marcello P. B. *Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção.* Ci. Inf., Brasília, v. 32, n. 3, p. 7-20, set./dez. 2003.

[75] Novello, T. C. *Ontologias, Sistemas baseados em Conhecimento e Modelos de Banco de Dados*. http://www.inf.ufrgs.br/~clesio/cmp151/cmp15120021/artigo_taisa.pdf. Acessado em: maio de 2005.

[76] Meneses, E. X. *Jornada de Atualização em Inteligência Artificial- Integração de Agentes de Informação.* 2001. http://www.ime.usp.br/~eudenia/jaia/. Acessado em novembro de 2004.

[77] Gudwin, R. R. *Introdução à Teoria dos Agentes*. DCA-FEEC-UNICAMP. http://www.dca.fee.unicamp.br/~gudwin/courses/IA009. Acessado em agosto de 2005.

[78] McCarthy, J., *Elephant 2000: A Programming Language Based on Speech Acts*, *unpublished manuscript*. Formal Reasoning Group. Stanford University http://www-formal.stanford.edu/jmc/elephant/elephant.html. Acessado em abril de 2005.

[79] Vasudevan, V. *Comparing Agent Communication Languages. Object Servces and Consulting*, Inc. Julho de 1998. http://www.objs.com/agility/tech-reports/9807-comparing-ACLs.html. Acessado em fevereiro de 2005.

A – A INTERNET E O TCP/IP

A ARPANET foi criada em 1969 pela agência de pesquisas DARPA (*Defense Advanced Research Projects Agency*) visando ser apenas uma rede para pesquisas e desenvolvimento de novas tecnologias de comunicação de dados procurando robustez, confiabilidade e, principalmente, a independência em relação ao meio físico usado, por meio de um padrão aberto para comutação de pacotes. Porém, seu sucesso foi tamanho que diversas organizações começaram a usá-la frequentemente, até que, em 1975, foi oficializada sua condição de operacional. Contudo, os investimentos em pesquisas, principalmente por órgãos militares como o *Department of Defense* (DoD), continuaram a fim de um maior avanço, resultando na concepção, padronização e adoção dos protocolos do TCP/IP em toda rede em 1983 [TANENBAUM, 1997].

No advento da padronização do protocolo TCP/IP, a grande ARPANET foi dividida em duas subredes: uma para fins militares, chamada de MILNET seria uma parte da *Defense Data Network* (DDN), e outra parte menor que continuou sendo chamada de ARPANET. O termo Internet começou a ser usado nessa época para se referir a grande rede formada pela MILNET com a ARPANET. Rapidamente, a Internet foi crescendo por meio da junção da ARPANET com grandes redes existentes na época como a NFSNET, uma rede com fins puramente científicos fundada pela NSF (*National Science Foundation*), além de diversas outras redes regionais, estaduais e continentais.

Em 1990, a Internet já conectava 3 mil redes e 200 mil computadores. Em 1992 1 milhão de computadores se conectavam à rede. Em 1995, a rede já conectava inúmeros *backbones*, centenas de redes regionais, milhares de redes locais, milhões de computadores e dezenas de milhares de usuários [PAXSON, 1994].

A adoção do TCP/IP naquele momento não ocorreu por acaso, nem por imposição dos órgãos militares. Seus protocolos atendiam exatamente a grande necessidade por comunicação de dados em nível global. Além disso, traziam diversas características que contribuíram para sua popularização como [HUNT, 1995]:

- Padrões abertos e livremente disponíveis que não dependiam dos sistemas operacionais ou *hardwares* específicos. Todos os padrões são definidos e publicados em consenso com a comunidade científica, industrial e entidades de

padronização de todo o mundo e, geralmente, são regidos em documentos com linguagem técnica e direta chamados de RFCs (*Requests for Comments*) que contém as últimas versões das especificações de todos protocolos do TCP/IP;

- Independência em relação à tecnologia do *hardware* da rede. O TCP/IP poderia operar em cima de diferentes tecnologias, como o *Ethernet*, *token ring*, X.25, ATM, o que favoreceu sua proliferação;
- Um sistema de endereçamento que permitia que um dispositivo de rede fosse unicamente identificado em toda a rede. Esse identificador foi nomeado mais tarde como endereço IP.

A.1 - ARQUITETURA DO PROTOCOLO TCP/IP

A arquitetura do protocolo TCP/IP pode ser tratada como uma simplificação do modelo de referência OSI (*Open System Interconnect Reference Model*) [DAY & ZIMMERMANN, 1983] desenvolvida pela *International Standards Organization* - ISO que visa propor uma série de definições e terminologias para uso em qualquer protocolo de comunicação de rede. Ao invés de trazer as sete camadas do modelo OSI, o TCP/IP costuma ser representado com um número entre três a cinco camadas. Na presente abordagem, o modelo será representado com quatro camadas, sendo uma aproximação com enfoque mais prático, presente nos principais manuais [HUNT, 1992], como mostrado na Figura A.1.

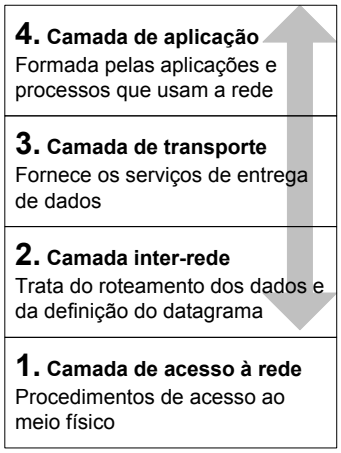


Figura A.1 - Camadas do TCP/IP

Cada camada da pilha TCP/IP é responsável por uma função distinta na comunicação. Para que um usuário possa enviar uma mensagem qualquer pela rede, é preciso que esse dado passe por todas as camadas até que encontre o meio físico da rede. O mecanismo oposto vale para o recebimento de mensagens.

Como mostrado na Figura A.2, quando os dados descem as camadas, ou seja, são enviados, são adicionadas informações de controle para o futuro tratamento de cada camada. Essas informações de controle são adicionadas sobre a forma de cabeçalhos (ou

headers) que são anexados antes de sua área de dados. O cabeçalho somado a área de dados de uma camada é completamente encapsulado pela camada inferior que considerará todo conteúdo recebido como sendo sua nova área de dados. Ou seja, na camada mais baixa, todos os cabeçalhos e dados das camadas superiores serão considerados como dados.

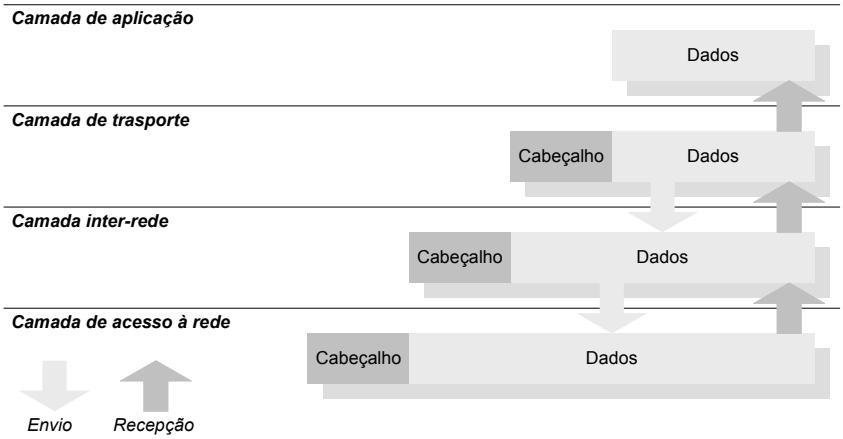


Figura A.2 - Encapsulamento de dados no TCP/IP

Na recepção de dados da rede, ocorre o mecanismo oposto. Os cabeçalhos são extraídos por cada camada e após o tratamento necessário são enviados para a camada superior que receberá apenas o que ela necessita. Após o processamento de todas as camadas, a informações estará em sua forma bruta e pronta para a leitura pela respectiva aplicação.

A.2 - CAMADA DE ACESSO À REDE

A camada de acesso à rede, mostrada na Figura A.1 (1), é responsável pela transmissão dos dados da camada de inter-rede ou datagramas, usando o dispositivo de rede instalado, por isso os protocolos dessa camada são específicos para cada hardware. Algumas das funções dessa camada são o encapsulamento dos datagramas em formato apropriado para serem transmitidos pela tecnologia de rede disponível, e o mapeamento do endereço IP para o endereço físico usado pela rede. Um exemplo de protocolo dessa camada é o ARP (*Adress Resolution Protocol*) [RFC826] que especificam a conversão de endereços IP para endereços MAC do Ethernet e vice-e-versa.

A.3 - CAMADA INTER-REDE

A camada de inter-redes, mostrada na Figura A.1 (2), é chamada informalmente de camada IP, devido ao seu protocolo mais importante ser o *Internet Protocol* [RFC791]. Essa camada é responsável, por meio do protocolo IP, pelas funções de definição do datagrama (unidade básica de transmissão da Internet também chamado de pacote IP), definição do endereçamento, movimentação dos dados entre a camada de acesso à rede e a camada de transporte, roteamento dos datagramas e pelos mecanismos de fragmentação e desfragmentação dos datagramas.

Em relação ao roteamento dos pacotes, um datagrama IP possui todas as informações necessárias para que possa trafegar em uma rede TCP/IP e atingir seu destinatário. O processo de roteamento é viabilizado por dois componentes básicos: os roteadores (ou *gateways*) e os *hosts*. Os roteadores são os dispositivos que repassam os pacotes para outras redes com base em uma tabela de rotas. Para isso, não precisam enxergar as camadas superiores, mas, apenas até a camada de inter-rede, onde existem as definições do datagrama IP. Já os *hosts* são os dispositivos que fazem o envio ou a recepção dos dados, exigindo que haja o processamento de todas as camadas para que seja atingida a aplicação.

O formato de um datagrama IP é mostrado na Figura A.3 e sucintamente descrito pela Tabela A.1 com base nas definições encontradas em [TANENBAUM, 1997].

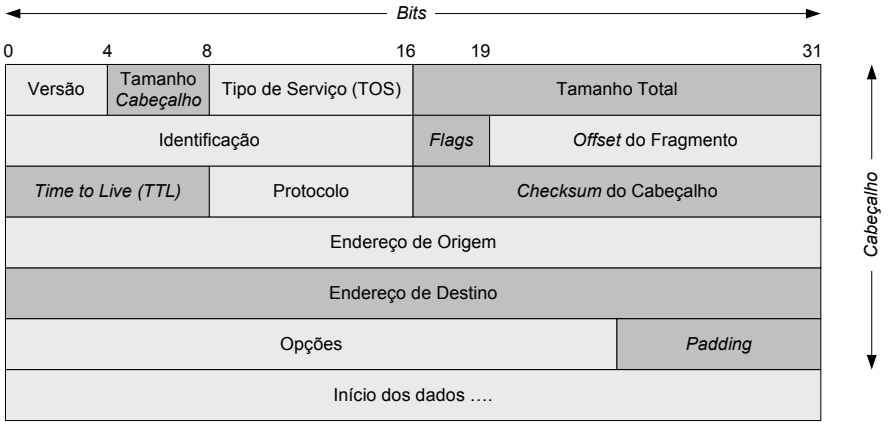


Figura A.3 - Formato do datagrama IP

Tabela A.1 - Formato do datagrama IP

Campo	***Descrição***	***Tam. (bits)***
Versão	Versão do protocolo a que o datagrama pertence. Usada em caso de transições entre versões.	4
Tam. do Cabeçalho (IHL)	Define o tamanho do cabeçalho do datagrama, pois o campo de opções pode estar preenchido.	4
Tipo de Serviço (TOS)	Possibilita a definição de níveis de qualidade de serviço para a rede que transmitirá esse datagrama. Alguns parâmetros são: confiabilidade, retardo e velocidade. Na prática esses campos são ignorados.	8
Tamanho total	Tamanho do datagrama completo formado pelo cabeçalho e os dados. O tamanho máximo é 65535 bytes.	16
Identificação	Em caso de fragmentação dos datagramas permite a identificação das partes por meio de um número comum.	16
Flags	Fazem o controle dos fragmentos por meio de estados do tipo DF e MF. O DF proíbe a fragmentação desse datagrama. O MF avisa que o datagrama ainda terá mais fragmentos.	3
Offset do Fragmento	Identifica em qual posição no datagrama o atual fragmento deve ser colocado.	13
Time to Live (TTL)	Define o tempo de vida de um pacote. Caso esse tempo ultrapasse, o pacote será descartado no próximo. O valor máximo é de 256 segundos.	8
Protocolo	Define qual o protocolo de transporte que será usado pela camada de transporte.	8
Checksum do cabeçalho	Serve para a verificação de integridade do cabeçalho do datagrama.	16
Endereço IP de origem	Define a rede de origem e a máquina de origem do datagrama. Os endereços IP geralmente são representados em notação decimal separa por pontos para cada octeto. Por exemplo: 200.175.180.46 (11001000 10101111 10110100 0101110).	32
Endereço IP de destino	Define a rede de destino e a máquina de destino para esse datagrama.	32
Opções	Permite que novas implementações sejam possíveis no cabeçalho.	24

Padding	Apenas um espaçamento que também pode ser usado por aplicações opcionais.	8

Além do protocolo IP, outro também presente na camada de inter-redes é o protocolo ICMP (*Internet Control Message Protocol*) [RFC792]. O protocolo ICMP usa o serviço de entrega de datagramas da camada para enviar mensagens de controle, e erros relativos à Internet, por isso, muitas vezes é usado para a execução de testes. Uma mensagem ICMP é sempre encapsulada em um pacote IP para seu envio. Na Figura A.4, está representado o formato de um pacote ICMP.

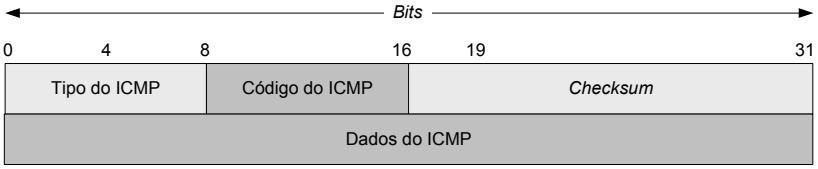


Figura A.4 - Formato de uma mensagem ICMP

Os campos de tipo e código do ICMP definem qual mensagem de controle ou erro está sendo transmitida. Algumas mensagens comuns são listadas na Tabela A.2 [RFC792].

Tabela A.2 - Alguns tipos de mensagens ICMP

Tipo da mensagem	***Descrição***
`Destination unreachable`	O pacote não pôde atingir seu destino.
`Destination port unreachable`	A porta de serviço não foi encontrada no destino.
`Time exceeded`	O campo do pacote TTL expirou.
`Parameter problem`	Existência de algum campo inválido no cabeçalho.
`Echo request`	Consulta se uma máquina está ativa na rede.
`Echo reply`	Reposta afirmativa à consulta de máquina ativa.

A.4 - CAMADA DE TRANSPORTE

A camada de transporte, mostrada na Figura A.1 (3), tem a função básica de oferecer os serviços de comunicação fim-a-fim entre as aplicações [SOARES, LEMOS & COLCHER, 1995]. Os principais protocolos presentes nessa camada são o *Transmission Control Protocol* (TCP) e o *User Datagram Protocol* (UDP).

O TCP oferece um serviço confiável de comunicação dos dados de forma orientada à conexão com mecanismos de detecção e correção de erros, mesmo sobre redes que não possuem esses recursos. Para isso, o protocolo adiciona aos dados provenientes da camada de aplicação um cabeçalho com algumas informações de controle como mostrado na Figura A.5. Geralmente um pacote TCP é denominado como um segmento, visto que as aplicações que exigem conexão enviam uma seqüência de fluxos de dados (ou *streams*) respeitando a MTU (*Master Transfer Unit*).da rede. A Tabela A.3 detalha a função de cada campo.

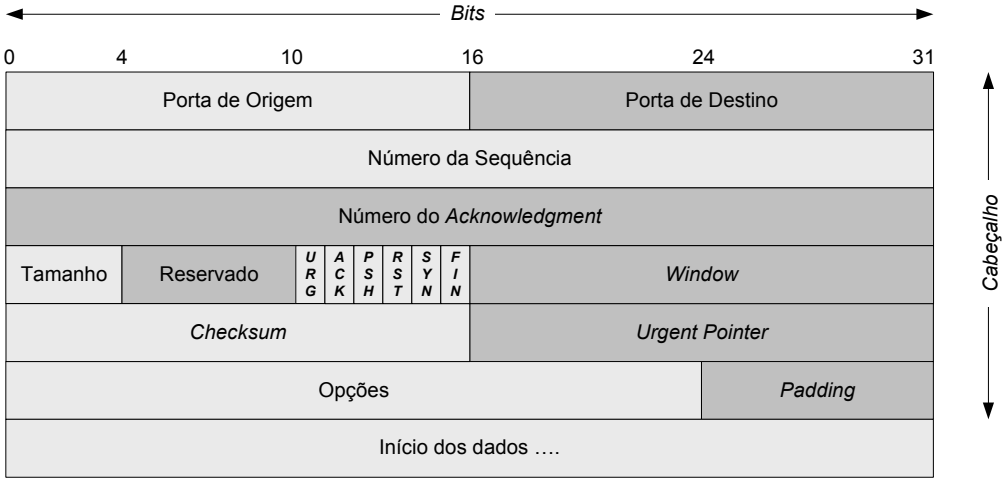


Figura A.5 - Formato de um segmento TCP

Tabela A.3 - Campos de um segmento TCP

Campo	***Descrição***	***Tam. (bits)***
Porta de Origem	Número que identifica a qual processo, na origem, a conexão está associada. Geralmente, serviços conhecidos já fixam os valores da portas, por exemplo, o FTP que se associa à porta 21.	16
Porta de Destino	Número que identifica a qual processo, no destino, a conexão está associada.	16
Número da Seqüência	Número seqüencial usado para o controle da ordenação dos segmentos.	32
Número do Acknowledgment	Número que representa o número de seqüência do último segmento recebido. Serve para controlar o fluxo e para realizar o *positive aknowledgment*.	32
Tamanho do cabeçalho	Define o tamanho do cabeçalho do datagrama, pois o campo de opções pode estar preenchido.	4

Flags		6
Window		16
Checksum		16
Urgent Pointer		16
Opções	Permite que novas implementações sejam possíveis no cabeçalho.	24
Padding	Apenas um espaçamento que também pode ser usado por aplicações opcionais.	8

A confiabilidade do TCP é garantida pelo mecanismo chamado de *Positive Acknowledgment with Re-transmission* (PAR). Esse mecanismo é bastante simples, os dados são reenviados caso não receba uma confirmação do recebimento pelo destinatário. Também com o uso do campo de *checksum*, é possível detectar corrupções nos dados e solicitar a re-transmissão por meio de um *positive acknowledgment*.

Para estabelecer uma conexão fim-a-fim entre os *hosts*, o protocolo TCP inicia a comunicação com um período de negociação e, após isso, começa a transferência dos dados. A metodologia usada nesse preâmbulo da conexão é chamada de *three-way handshake*, pois sempre envolve três segmentos (ou pacotes TCP) para sua realização, conforme mostrado na Figura A.6.

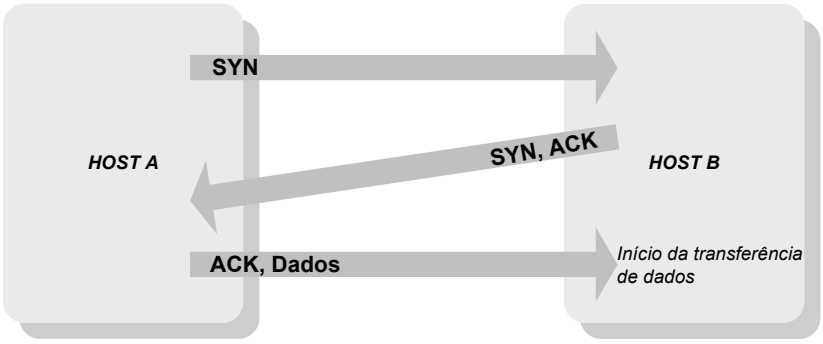


Figura A.6 - Metodologia de negociação Three-way Handshake

Para iniciar a conexão com o regime do *three-way handshake* , o processo se dá do seguinte modo:

- O *host* A envia para o *host* B um segmento com o bit SYN ("*Synchronize sequence number"*) marcado e um número de seqüência definido, significando que deseja iniciar a uma conexão com o *host* B por meio de um segmento marcado com aquele número de seqüência;
- Caso queira prosseguir, o *host* B reponde ao *host* A com um segmento cujos bits ACK ("*Acknowledgment"*) e SYN estão marcados, informando, também, com que número de seqüência irá iniciar. Caso queira rejeitar essa conexão, deverá enviar um segmento com o bit RST ("*Reset"*) marcado;
- Finalmente o *host* A envia um segmento com o bit ACK, indicando que o número de seqüência é conhecido, e já inicia a transferência dos dados. Para finalizar a conexão o *host* A deverá enviar um segmento com o bit FIN ("*No more data from sender"*) marcado.

O UDP, por sua vez, opera sem conexão, fornecendo um serviço não confiável para entrega de datagramas, porém extremamente simples e com baixo *overhead* de protocolo. Sua não confiabilidade é devida a inexistência de técnicas que certificam o recebimento das mensagens em seu destino na rede. A Figura A.7 denota o formato de uma mensagem UDP.

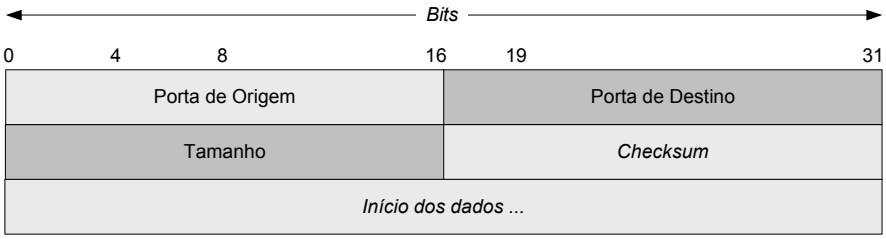


Figura A.7 - Formato da mensagem UDP

Da mesma maneira que no TCP, os campos relativos às portas de origem e destino servem para o mapeamento da entrega de dados com as aplicações. As aplicações que se baseiam em pequenas consultas e respostas, como o DNS (*Domain Name System*), dão preferência ao uso do UDP devido à sua eficiência, pois em caso do não recebimento das mensagens, custa menos reenviá-los do que implementar algum tipo de controle de erro.

A.5 - CAMADA DE APLICAÇÃO

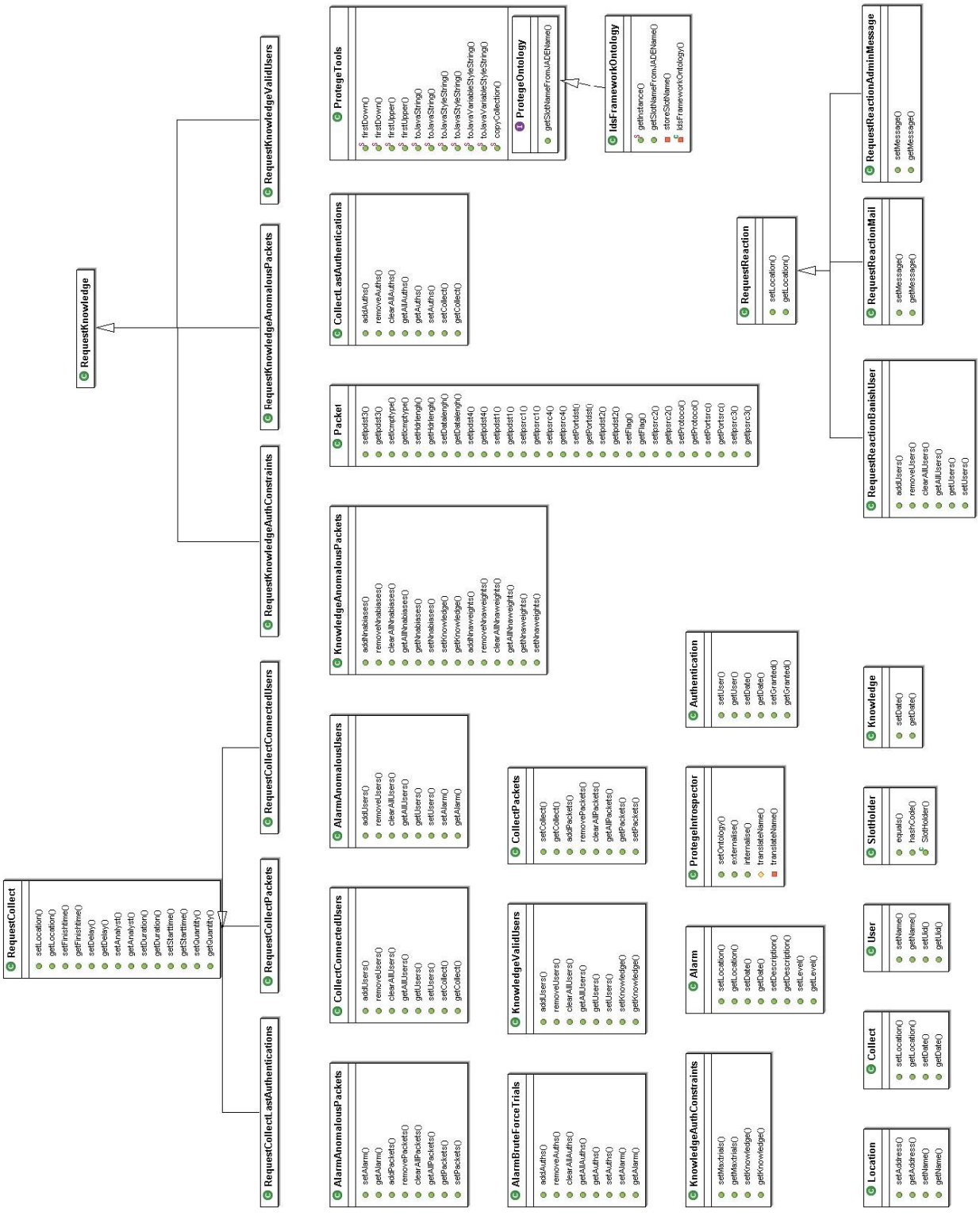
Na camada de aplicação, mostrada na Figura A.1 (4), residem os processos das aplicações que usam os serviços de entrega da camada de transporte. As aplicações, em sua grande maioria, são serviços de rede ou do usuário que usam os protocolos TCP e UDP conforme suas necessidades. A Tabela A.4 lista algumas aplicações conhecidas e seus respectivos protocolos de transporte e RFCs.

Tabela A.4 - Exemplos de protocolos de aplicação

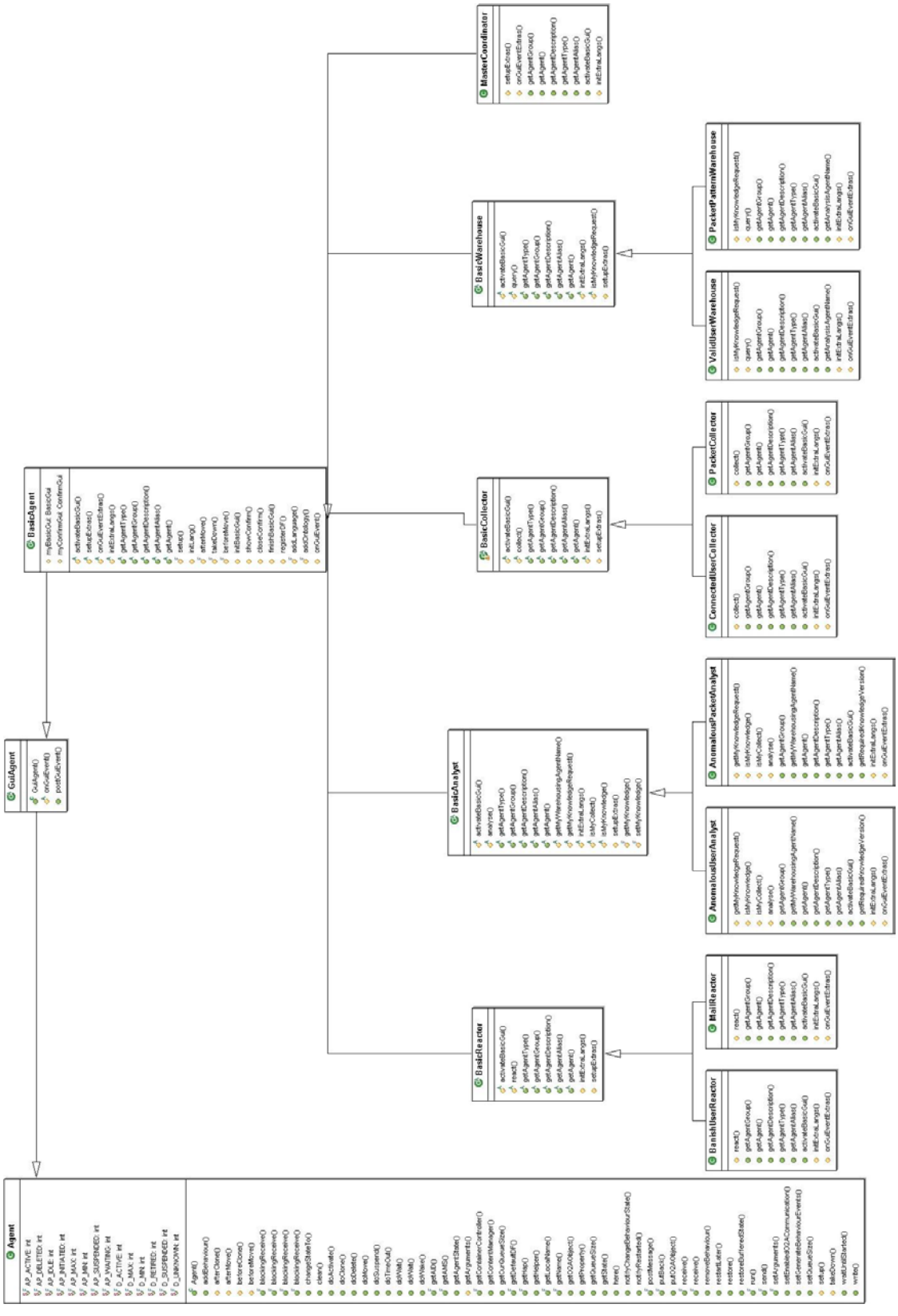
Protocolo de aplicação	***RFC***	***Protocolo de transporte***
FTP	RFC-959	TCP [RFC793]
Telnet	RFC-854	TCP
SMTP	RFC-821	TCP
DNS	RFC-1035	UDP
NFS	RFC-1094	UDP

B – DIAGRAMAS DE CLASSES

B.1. DIAGRAMA DE CLASSES DAS ONTOLOGIAS



B.2. DIAGRAMA DE CLASSES DO FRAMEWORK



B.3. DIAGRAMA DAS CLASSES DOS COMPONENTES DE SERVIÇOS