Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-graduação em Engenharia de Eletricidade

Lindonete Gonçalves Siqueira

Tolerância a Falhas para o NIDIA: um Sistema de Detecção de Intrusão Baseado em Agentes Inteligentes

São Luís
2006

Lindonete Gonçalves Siqueira

Tolerância a Falhas para o NIDIA: um Sistema de Detecção de Intrusão Baseado em Agentes Inteligentes

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para obtenção do título de Mestre em Engenharia de Eletricidade, Àrea de Concentração: Ciência da Computação.

Orientador: Zair Abdelouahab

Doutor em Ciência da Computação - UFMA

São Luís

2006

Siqueira, Lindonete Gonçalves

Tolerância a Falhas para o NIDIA: um Sistema de Detecção de Intrusão Baseado em Agentes Inteligentes / Lindonete Gonçalves Siqueira - São Luís, 2006.

104 f.

Dissertação (Mestrado) - Universidade Federal do Maranhão - Programa de Pós-graduação em Engenharia de Eletricidade.

Orientador: Zair Abdelouahab.

1.Tolerância a Falhas 2. Sistema Multiagente 3. Sistema - Confiabilidade. I.Título.

CDU 004.052.3

Lindonete Gonçalves Siqueira

Tolerância a Falhas para o NIDIA: um Sistema de Detecção de Intrusão Baseado em Agentes Inteligentes

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para obtenção do título de Mestre em Engenharia de Eletricidade, Àrea de Concentração: Ciência da Computação.

Aprovado em 10 de julho de 2006

BANCA EXAMINADORA

Zair Abdelouahab

Doutor em Ciência da Computação - UFMA

Denivaldo Cícero Pavão Lopes

Doutor em Ciência da Computação - UFMA

Djamel Fawzi Hadj Sadok

Doutor em Ciência da Computação - UFPE

Aos meus pais Lauriano e Lindalva.

Agradecimentos

A Deus pela proteção e bênçãos recebidas.

Aos meus pais, Lauriano e Lindalva, pela torcida, compreensão, carinho e preocupação.

Aos meus irmãos, Lindovânia, Laurinete, Júnior e Leonardo pelo apoio incondicional.

Ao professor Zair Abdelouahab pela orientação.

Aos meus companheiros do mestrado, Mauro e Simone, pelas incansáveis horas de estudo, inclusive aos finais de semana.

A todos os meus amigos: os novos (Osvaldo, Ricardo, Clissiane, Jonhneth, Adriano, Irlandino, Francislene, Raimundo, Rafael, Bismark, Francisco, Cícero, Emerson, Emanuel) ou antigos; os que acompanharam de perto ou de longe.

Ao CNPq pela bolsa e à FAPEMA pelo apoio na publicação do artigo.

Por fim, a todos que direta ou indiretamente contribuiram para a elaboração deste trabalho.

"Não sabendo que era impossível,
foi lá e fez."
Paulo Leminsk

Resumo

Entre as diversas ferramentas existentes para prover segurança a um sistema computacional destaca-se o Sistema de Detecção de Intrusão (SDI). O SDI tem como objetivo identificar indivíduos que tentam usar um sistema de modo não autorizado ou que têm autorização, mas abusam dos seus privilégios. Porém, um SDI para realizar corretamente sua função precisa, de algum modo, garantir confiabilidade e disponibilidade a sua própria aplicação. Portanto, o SDI deve dar continuidade aos seus serviços mesmo em caso de falhas, principalmente falhas causadas por ações maliciosas. Esta dissertação propõe um mecanismo de tolerância a falhas para o Projeto *Network Intrusion Detection System based on Intelligent Agents* (NIDIA), um sistema de detecção de intrusão baseado na tecnologia de agentes. O mecanismo utiliza duas abordagens: o monitoramento do sistema e a replicação de agentes. O mecanismo possui uma sociedade de agentes que monitora o sistema para coletar informações relacionadas aos seus agentes e *hosts* e para prover uma recuperação adequada para cada tipo de falha detectada. Usando a informação que é coletada, o sistema pode: descobrir os agentes não ativos; determinar quais os agentes que devem ser replicados e qual estratégia de replicação deve ser usada. A estratégia de replicação depende do tipo de cada agente e da importância do agente para o sistema em diferentes momentos do processamento. Além disso, esse monitoramento também permite realizar outras importantes tarefas tais como balanceamento de carga, migração, e detecção de agentes maliciosos, para garantir a segurança do próprio SDI (*self protection*). A implementação da arquitetura proposta e os testes realizados demonstram a viabilidade da solução.

Palavras-chave: Segurança, Detecção de Intrusão, Tolerância a Falhas, Sistema Multiagente, Confiabilidade.

Abstract

An Intrusion Detection System (IDS) is one tool among several existing ones to provide safety to a computational system. The IDS has the objective of identifying individuals that try to use a system in non-authorized way or those that have authorization but are abusing of their privileges. However, to accomplish the functions correctly an IDS needs to guarantee reliability and availability of its own application. The IDS should provide continuity to its services in case of faults, mainly faults caused by malicious actions. This thesis proposes a fault tolerance mechanism for the Network Intrusion Detection System based on Intelligent Agents Project (NIDIA), an intrusion detection system based on the agents technology. The mechanism uses two approaches: monitoring the system and replication of agents. The mechanism has a society of agents that monitors the system to collect information related to its agents and *hosts* and to provide an appropriate recovery for each type of detected fault. Using the information that is collected, it is possible: to discover agents that are not active; determine which agents must be replicated and which replication strategy must be used. The replication type depends on the type of each agent and its importance for the system in different moments of processing. Moreover, this monitoring allows to accomplish other important tasks such as load balancing, migration, and detection of malicious agents, to guarantee safety of the proper IDS (*self protection*). The implementation of the proposed architecture and the illustrated tests demonstrate the viability of the solution.

Keywords: Security, Intrusion Detection, Fault Tolerance, Multiagent System, Reliability.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AAFID	Autonomous Agents For Intrusion Detection
ACL	Agent Communicaton Language
AMS	Sistema gerenciador de agentes (Agent Management System)
BA	Agente BAM
BAM	Binary Association Memory
CIDF	Common Intrusion Detection Framework
DARPA	Defense Advanced Research Projects Agency
DARX	Dynamic Agent Replication eXtension
DF	Facilitador de diretórios (Directory Facilitator)
DFDB	Base de Dados de Incidentes de Intrusão e Informação Forense
FIPA	Foundation for Intelligent Physical Agents
HNA	Agente Honey Net
IIDB	Base de Dados de Padrões de Intrusos e Intrusões
JADE	Java Agent DEvelopment Framework
JVM	Java Virtual Machine
LSIA	Agente de Segurança Local (Local Security Intelligent Agent)
MADIDS	Mobile Agent Distributed Intrusion Detection System
MCA	Agente Controlador Principal
MLP	Perceptron de Múltiplas Camadas (Multilayers Perceptron)
MTP	Message Transport Protocol
NIDIA	Network Intrusion Detection System based on Intelligent Agents
OKDB	Base de Dados Otimizada de Palavras-chave
PRDB	Base de Dados de Perfis (Profile Database)
RADB	Base de Dados de Ações de Respostas
RMA	Remote Management Agent
RMI	Remote Method Invocation
SAARA	Sociedade Atualizada de Agentes de Reconhecimento de Assinaturas
SCA	Agente Controlador de Ações (System Controller Agent)
SDI	Sistema de Detecção de Intrusão

SEA	Agente de Avaliação de Segurança do Sistema (System Security Evaluation Agent)
SFEA	Agente de Avaliação de Falhas (System Fault Evaluation Agent)
SFTA	System Fault Tolerance Agent
SIA	Agente de Integridade do Sistema (Self-Integrity Agent)
SMA	Agente de Monitoramento do Sistema (System Monitoring Agent)
SRA	Agente de Replicação (System Replication Agent)
SSA	Agente Sentinela (System Sentinel Agent)
SSL	Secure Socket Layer
STDB	Base de Dados de Estratégias
SUA	Agente de Atualização do Sistema (System Updating Agent)
TCP	Transmission Control Protocol
UFMA	Universidade Federal do Maranhão
UML	Unified Modeling Language
XML	eXtensible Markup Language
WS-RM	Web Service -Reliable Messaging

Lista de Códigos

Sumário

1 Introdução

A rede mundial de computadores vem apresentando grande aumento no número de ataques, isto é, tentativas de comprometer a integridade e a confiabilidade dos sistemas [Santos and Campello, 2001]. E manter organizações longe de ataques é um desafio cada vez maior para evitar furto e alteração de informações ou até mesmo a paralisação de sistemas.

As ferramentas voltadas para explorar as vulnerabilidades[1] dos sistemas vêm aumentando em complexidade e eficiência, facilitando cada vez mais a tarefa dos atacantes. Os *hackers*, com conhecimentos técnicos em ciência da computação e usando ferramentas pré-definidas e disponibilizadas na Internet, rompem poderosos sistemas de segurança das mais diversas empresas. Além disso, alteram dados, operam transferências de recursos à ordem de milhões de dólares, sem que para isto precisem sacar uma arma, ou sair detrás de um simples microcomputador [Security, 2005].

Para se ter uma idéia do crescimento dos incidentes de segurança, entre 1999 e 2004 houve um aumento de 2.000% nas notificações de incidentes relatados ao CERT.br, Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil [CERT.br, 2005]. Em 1999 foram relatados 3.107 incidentes e em 2004 chegou-se a um total de 75.722 incidentes, como pode ser observado na Figura 1.1. As estatísticas[2] mostram que em 2005 até o mês de setembro foi relatado um total de 46.205 incidentes de segurança.

Isto mostra a necessidade de ferramentas que auxiliem as organizações a manter a segurança de suas informações. Entre essas ferramentas destaca-se o *firewall*, o antivírus e o Sistema de Detecção de Intrusão (SDI) [Ramachandran and Hart, 2004], que, preferencialmente, combinados tornam o sistema mais robusto contra incidentes.

[1] A vulnerabilidade é qualquer fraqueza que possa ser explorada para atacar um sistema.

[2] Estas notificações são voluntárias e refletem os incidentes ocorridos em redes que espontaneamente notificaram ao CERT.br.

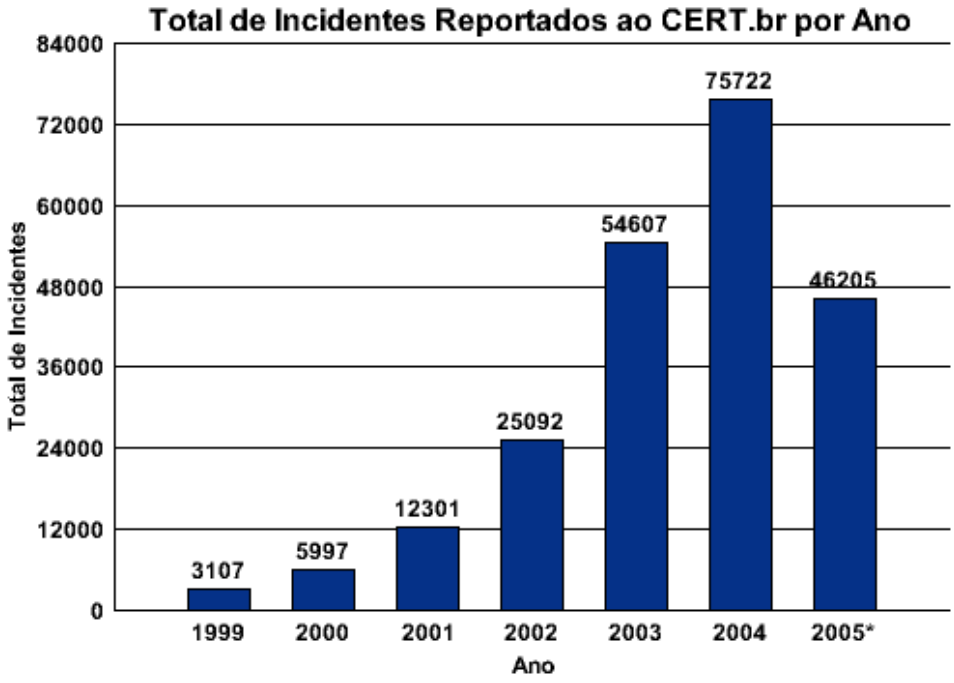

Figura 1.1: Incidentes relatados ao Cert.br. (* até 09/2005)

1.1 Descrição do Problema

Detecção de intrusão é definida como a identificação de indivíduos que tentam usar um sistema de modo não autorizado ou que têm autorização, mas abusam dos seus privilégios [Balasubramaniyan et al., 1998, Hegazy et al., 2003]. Os sistemas de detecção de intrusão são ferramentas de segurança que auxiliam o administrador de rede a prevenir ataques e a tomar uma ação quando um ataque é iniciado ou detectado.

No entanto, segundo [Campello et al., 2001] "[...] um dos principais responsáveis pela segurança de uma organização, o sistema de detecção de intrusão vem se tornando alvo freqüente dos atacantes mais experientes, comprometido por técnicas de negação de serviço (*denial-of-service*[3]), *spoofing*[4], dentre outras[...]".

Portanto um SDI precisa, de algum modo, garantir confiabilidade e disponibilidade à sua própria aplicação, de tal forma que consiga dar continuidade aos seus serviços mesmo em caso de falhas, principalmente as falhas causadas por ações

[3]Segundo definição do CERT/CC (Computer Emergency Response Team Coordination Center) um ataque de "negação-de-serviço" é caracterizado por uma tentativa explícita de impedir que os usuários legítimos de um serviço usem esse serviço. Esse ataque pode desabilitar um computador ou até mesmo uma rede.

[4]*spoofing* é o ato de falsificar o remetente de um pacote de transmissão de dados.

maliciosas [Santos and Campello, 2001]. A confiabilidade pode ser definida como sendo a capacidade que um sistema tem em se manter funcionando. Um sistema deve continuar comportando-se de acordo com sua especificação independente do tipo de falha, seja por problemas físicos de *hardware*, de implementação ou mesmo por ações mal intencionadas desencadeadas por algum atacante. A disponibilidade é a probabilidade de que o sistema esteja funcionando em um dado instante.

Existem, atualmente, inúmeros esforços de pesquisa nesse sentido, demonstrando a importância dessas características em qualquer tipo de sistema. No entanto, isto ainda não é uma realidade em sistemas de detecção de intrusão [Campello et al., 2001].

Uma solução para o problema citado acima é aplicar a tolerância a falhas nos sistemas de detecção de intrusão de modo a levá-los a um estado mais seguro e resistente a falhas. A tolerância a falhas representa todo um conjunto de técnicas e ações empregadas na tentativa de minimizar as falhas existentes e suas conseqüências. E assim, garantir um funcionamento contínuo e confiável por parte dos sistemas computacionais, ou seja, a continuidade do serviço deve ser possível mesmo na presença de falhas.

1.2 Objetivos Gerais e Específicos

Na Universidade Federal do Maranhão (UFMA), o Projeto *Network Intrusion Detection System based on Intelligent Agents* (NIDIA) proposto por [Lima, 2002] tem sido desenvolvido. O NIDIA é estruturado como uma sociedade de agentes inteligentes e tem como importante característica o fato de permitir detecção de ataques novos usando técnicas de redes neurais e a capacidade de dar uma resposta aos ataques.

Esta dissertação propõe um mecanismo de tolerância a falhas ao NIDIA. Este mecanismo deve ser capaz de detectar falhas acidentais e maliciosas. O mecanismo é composto por uma sociedade de agentes que monitoram o sistema e o recuperam quando uma falha é detectada. Agentes, usando o conceito de sentinelas [Haegg, 1997], monitoram o sistema para coletar informações relacionadas aos agentes e aos *hosts* (onde os agentes do NIDIA estão executando). Baseado nas informações que são coletadas é possível: detectar que agentes estão ativos; detectar que agentes devem ser replicados e que estratégia de replicação deve ser usada. O processo de replicação depende do tipo de cada agente e da importância deste agente ao sistema em diferentes momentos do processamento.

Além disso, com o monitoramento realizado por agentes (com função de sentinela) pode-se realizar ainda algumas tarefas importantes ao sistema tais como: balanceamento da carga, migração dos agentes e detecção de agentes maliciosos, garantindo, assim, a segurança do próprio sistema de detecção (autoproteção).

Esta dissertação tem por objetivo:

- Apresentar um mecanismo de tolerância a falhas ao NIDIA.
- Apresentar um protótipo do mecanismo de tolerância a falhas por meio de uma sociedade de agentes inteligentes que monitoram e recuperam o sistema de forma automática.

1.3 Estrutura da Dissertação

Esta dissertação encontra-se organizada em 7 capítulos, descritos a seguir:

No capítulo 1, os problemas atuais de segurança enfrentados pelos sistemas de detecção de intrusão são apresentados. Também os objetivos gerais e específicos e a organização desta dissertação são discutidos.

No capítulo 2, uma visão geral sobre sistemas multiagentes e sistemas de detecção de intrusão é apresentada, destacando-se classificação, características e projetos de sistemas de detecção que são baseados em agentes. Por fim, informações sobre mecanismos de tolerância a falhas são discutidas, por exemplo, como e onde são aplicados esses mecanismos em sistemas multiagentes.

No capítulo 3, o *Network Intrusion Detection System based on Intelligent Agents* (NIDIA) é apresentado. A arquitetura antiga e a atual e aspectos particulares do NIDIA são mostrados.

No capítulo 4, o mecanismo proposto para prover tolerância a falhas ao NIDIA é apresentado. A arquitetura do mecanismo baseada em agentes, a funcionalidade de cada agente e a integração entre eles e o NIDIA são apresentados. Alguns cenários para exemplificar seu funcionamento são mostrados.

No capítulo 5, aspectos da ferramenta utilizada para implementar o mecanismo, detalhes da implementação da arquitetura do mecanismo e do NIDIA e os

protótipos dos agentes que compõem o mecanismo são apresentados. Finalmente, alguns resultados parciais do protótipo obtidos nas simulações realizadas em laboratório são apresentados.

O capítulo 6 mostra as conclusões obtidas no desenvolvimento deste trabalho. Possíveis trabalhos futuros, que podem ser realizados a partir deste, são apresentados também. Por fim, as considerações finais desta dissertação são apresentadas.

2 Fundamentos Teóricos

Neste capítulo, alguns conceitos importantes sobre agentes e sistemas multiagentes, sistemas de detecção de intrusão, sistemas de detecção de intrusão baseados em agentes e, por fim, conceitos de tolerância a falhas são apresentados. Estes conceitos foram de extrema importância para o desenvolvimento do mecanismo de tolerância a falhas proposto nesta dissertação.

2.1 Agentes e Sistemas MultiAgentes

2.1.1 Agentes

De acordo com [Hegazy et al., 2003], um agente é um módulo de programa que executa continuamente em um ambiente particular. Ele pode realizar atividades de uma maneira flexível e inteligente e que possa responder às mudanças no ambiente. Um agente é autônomo e capaz de aprender com suas experiências. Ele executa ações baseando-se em seu conhecimento interno e em suas experiências passadas.

Características

A seguir, destacamos algumas características importantes dos agentes [Yepes, 2005, Karlsson et al., 2005]:

- Autonomia: um agente pode operar sem a intervenção de humanos ou de outros agentes. Ele tem controle sobre seu estado interno e comportamento. Um agente pode ou não necessitar de dados produzidos por um outro agente, mas ainda sim é considerado autônomo;
- Sociabilidade: um agente é capaz de interagir/cooperar com outros agentes (humanos ou não) por meio de uma linguagem de comunicação entre agentes;
- Reatividade: um agente é capaz de perceber estímulos do seu ambiente e reagir a estes estímulos;

- Proatividade: um agente não é apenas uma entidade que reage a um estímulo, ele possui a habilidade de iniciar ações;
- Mobilidade: um agente possui a capacidade de locomover-se através de uma rede de computadores;
- Adaptabilidade: um agente possui capacidade de adaptação, ou seja, é capaz de alterar o seu comportamento com base na sua experiência (aprendizagem).

Classificação

Segundo [Hegazy et al., 2003], os agentes podem ser classificados em quatro categorias:

1. Agentes reflexivos simples *(simple reflex agents)*: eles percebem a entrada do ambiente deles e interpretam-na a um estado que combine suas regras;
2. Agentes que mantêm rastro do mundo *(agents that keep track of the world)*: eles mantêm um estado interno de entradas passadas já que suas ações necessitam ocorrer na correlação com os estados passados e os estados novos;
3. Agentes baseados em objetivos *(goal-based agents)*: eles necessitam saber alguma informação sobre seus objetivos porque os *percepts* (percepção do agente obtido usando os sentidos) não fornecem informação suficiente para que os agentes tomem a ação correta. Às vezes saber os objetivos não é suficiente para que os agentes façam a ação correta, especialmente quando há objetivos conflitantes;
4. Agentes baseados em utilidades *(utility-based agents)*: eles mapeiam os estados de percepção em números que determinam quão próximo os objetivos foram alcançados.

Categorias

A seguir algumas categorias de agentes:

Agentes Competitivos: são agentes que "competem" entre si para a realização de seus objetivos ou tarefas;

Agentes Coordenados ou Colaborativos: são agentes com a finalidade de alcançar um objetivo maior. Eles realizam tarefas específicas, porém coordenando-as

entre si de forma que suas atividades se completem;

Agentes Móveis: são agentes que tem a mobilidade como característica principal. Isto é, uma capacidade de mover-se seja por uma rede interna local (*intranet*) ou até mesmo pelo *Web*, transportando-se pelas plataformas levando dados e códigos. Os agentes móveis podem ser aplicados em: computação móvel, balanceamento de carga, comércio eletrônico, entre outras;

Agentes Estacionários: são aqueles opostos aos móveis. Isto é, são fixo em um mesmo ambiente e/ou plataforma;

Agentes Reativos: esses agentes realizam uma ação de acordo com outra ação efetuada com eles. Os agentes reativos se comportam segundo o modo estímulo-resposta, ou seja, não há uma memória sobre ações realizadas no passado e nem previsão de ações que poderão ser executadas no futuro;

Agentes Cognitivos: esses, ao contrário dos agentes reativos, podem raciocinar sobre as ações tomadas no passado e planejar ações a serem tomadas no futuro. Ou seja, um agente cognitivo é capaz de "resolver" problemas por ele mesmo. Ele tem objetivos e planos explícitos os quais permitem atingir seu objetivo final. Cada agente deve ter uma base de conhecimento disponível, que compreende todos os dados e todo o "*know-how*" para realizar suas tarefas e interagir com outros agentes e com o próprio ambiente. Além disso, devido a sua capacidade de raciocínio baseado nas representações do mundo, são capazes de ao mesmo tempo memorizar situações, analisá-las e prever possíveis reações para suas ações.

2.1.2 Sistemas MultiAgentes

Sistemas MultiAgentes são sistemas altamente distribuídos, ou seja, não possuem um único ponto de controle [da Silva, 2005]. Eles se baseiam na existência de uma sociedade composta por vários agentes que atuam no sistema por meio de cooperação e/ou concorrência. Os agentes podem interagir para aprender ou trocar experiências [Hegazy et al., 2003].

A efetividade de sistemas multiagentes pode ser medida pelo número de aplicações nas quais estes podem ser utilizados. Aplicações típicas de agentes são *e-commerce*, administração de rede, processamento e recuperação de informação, turismo

digital, sistemas de apoio, *web services*, médicas, grade e etc [Khan et al., 2005].

2.2 Sistemas de Detecção de Intrusão

Sistema de Detecção de Intrusão (SDI) é uma ferramenta utilizada para reforçar a política de segurança de uma empresa. Ele tem como principal objetivo identificar indivíduos que tentam utilizar um determinado sistema de forma não autorizada ou que desejam abusar de privilégios concedidos aos mesmos [Santos and Campello, 2001].

Basicamente, um sistema de detecção de intrusão é composto por três componentes que têm as funções de coletar, analisar e apresentar informações sobre o sistema [Allen et al., 1999]:

- Sensor: tem como objetivo coletar as informações do sistema;
- Analisador: componente principal de um SDI. Sua função é analisar os dados anteriormente obtidos e, através da análise, indicar se ocorreram ou não intrusões;
- Interface com o usuário: deve apresentar os dados coletados e analisados de forma organizada para o administrador.

2.2.1 Características

Em [Balasubramaniyan et al., 1998], algumas características importantes de um SDI são descritas:

- Deve rodar continuamente com o mínimo de intervenção humana. Deve permitir sua operação em *background*, mas deve ser de fácil compreensão e operação;
- Deve ser tolerante a falhas, de forma a não ser afetado por falhas no sistema, e sua base de conhecimento não deve ser perdida quando o sistema for reinicializado;
- Deve resistir a tentativas de mudança (subversão), ou seja, deve monitorar a si próprio de forma a garantir sua segurança e detectar se ele foi modificado por atacantes;

- Deve ter o mínimo de impacto no funcionamento do sistema onde está executando;
- Deve ser capaz de ser configurado de acordo com as políticas de segurança do sistema que está sendo monitorado;
- Deve ser capaz de se adaptar as mudanças do sistema e comportamento dos usuários no decorrer do tempo;
- Deve poder ser escalável para monitorar um grande número de *hosts*;
- Deve permitir reconfiguração dinâmica, ou seja, a habilidade de reconfigurar o SDI sem ter que reinicializá-lo.

2.2.2 Classificação

A seguir algumas classificações dos sistemas de detecção de intrusão segundo [Lima, 2002], [Santos and Campello, 2001] e [Balasubramaniyan et al., 1998].

Quanto à arquitetura

Caracteriza os sistemas de acordo com a localização dos componentes:

- Centralizada: arquitetura formada por componentes de levantamento de dados e por analisadores. Ela possui a desvantagem de não ser escalável e possuir único ponto de falha;
- Hierárquica: arquitetura formada por uma camada de coleta de dados, uma camada de analisadores, uma camada de tratamento de dados relevantes e por fim um coordenador central responsável pela ação a ser tomada. Porém se uma camada falhar o sistema como um todo estará comprometido, tornando o sistema vulnerável. Esta arquitetura possui ainda o problema de ter as camadas como único ponto de falha;
- Distribuída: arquitetura que não possui um gerenciador central. Ela possui componentes autônomos e independentes, e, portanto é escalável; e não possui pontos únicos de falhas;

- Híbrida: mecanismos centralizados interagem com módulos distribuídos, aproveitando as vantagens de cada um, tentando chegar a um ponto de equilíbrio entre desempenho, simplicidade, abrangência e robustez.

Utilizando essa abordagem, a escalabilidade é limitada, uma vez que processar todas as informações em um único *host* implica um limite no tamanho da rede que pode ser monitorada. No entanto, coleta de dados distribuída pode causar problemas como tráfego de dados excessivos na rede.

Quanto ao local da fonte de informações

Distingue os SDIs de acordo com o local em que a entrada de informações a serem analisadas se encontra:

- Baseado em *host*: captura e analisa dados internos a um *host*. Eles são instalados em servidores para alertar e identificar ataques e tentativas de acesso indevido à própria máquina, sendo mais empregados nos casos em que a segurança está focada em informações contidas em um servidor;
- Baseado em redes: captura e analisa o tráfego da rede. Eles são instalados em máquinas responsáveis por identificar ataques direcionados a toda a rede, monitorando o conteúdo dos pacotes de rede e seus detalhes como informações de cabeçalhos e protocolos;
- Baseado em aplicação: examina o comportamento de um programa de aplicação, geralmente na forma de arquivos de *log*;
- Híbrido: algumas abordagens usam os dois tipos (rede e host) para aumentar a capacidade de detecção. De acordo com [Ramachandran and Hart, 2004], a maioria dos sistemas de detecção envolve análise baseada em rede e baseada em *host*.

Os sistemas de detecção baseados em rede podem monitorar diversos computadores simultaneamente. Todavia, sua eficácia diminui na medida em que o tamanho e a velocidade da rede aumenta, pela necessidade de analisar os pacotes mais rapidamente. Além disso, o uso de protocolos cifrados (baseados em *SSL - Secure Socket Layer*) torna o conteúdo dos pacotes opaco ao SDI. A velocidade da rede e o uso de

criptografia não são problemas para os sistemas de detecção baseados em *host*. Todavia, como esse sistema é instalado na própria máquina a monitorar, pode ser desativado por um invasor bem-sucedido [Laureano, 2004].

Quanto aos métodos de detecção

Categoriza o SDI segundo o modo em que os dados são analisados em busca de traços de intrusões:

- Anomalia (mudança de padrão): este método é realizado observando as mudanças de uso em relação ao padrão normal do sistema (mudanças no padrão de utilização e comportamento do sistema). Uma base de dados do comportamento da rede é utilizada. A partir desta base é que o sistema verifica o que é ou não permitido, e quando encontra algo fora do padrão gera o alerta. Esse tipo é mais complicado de configurar, pois é difícil estabelecer o que é padrão em uma rede com muitos usuários. Porém, ele é melhor para detectar ataques desconhecidos, ou seja, novos ataques. No entanto podem ocorrer elevadas taxas de falso positivo e necessidade de treinamento do sistema;

- Assinaturas (Abuso): este método trabalha procurando regras pré-estabelecidas no tráfego da rede. Quando é encontrado algum código na rede que esteja descrito em alguma regra, é gerado um alerta ou evento que permita uma ação defensiva. Uma coleção de técnicas é mantida em uma base de dados, intrusões são encontradas utilizando essa base de dados para comparação;

- Híbrida: combina as duas abordagens anteriores, buscando detectar ataques conhecidos e comportamentos anormais.

Quanto ao comportamento na detecção

Categoriza o SDI em função da resposta que o mesmo irá ter quando um ataque for descoberto. A resposta a ataques seria a capacidade de reconhecer uma atividade maliciosa e então realizar ações para bloqueá-la ou minimizar suas conseqüências:

- Passivo: o SDI gera apenas um alerta ou notificação sobre a intrusão;

- Ativo: uma reação de defesa é desencadeada pelo SDI. Esse modo de comportamento nos SDIs é perigoso porque a tecnologia, apesar de ter mais de duas décadas de desenvolvimento, é um pouco imatura e muitos usuários podem ser prejudicados pelo sistema.

Quanto a freqüência de utilização

Conceito relativo ao tempo em que o sistema é monitorado:

- Monitoração Contínua: o sistema funciona em tempo real, as informações sobre o sistema são coletadas e analisadas no mesmo instante;
- Análise Periódica: o sistema funciona de modo estático, ou seja, periodicamente retira informações do ambiente e faz a análise necessária.

2.2.3 Limitações

Apesar de toda pesquisa direcionada à área de detecção de intrusão, ainda existem inúmeros problemas enfrentados pelos SDIs. Um exemplo desses problemas é a crescente sofisticação dos ataques e das ferramentas utilizadas pelos atacantes. Isso obriga as ferramentas de detecção de intrusão a constantes atualizações e conseqüentemente aumenta a sua complexidade. Outro exemplo dos problemas ainda enfrentados pelos atuais SDIs é a falta de padrões amplamente aceitos pela área de sistemas de detecção, o que dificulta as interações entre sistemas existentes e a correlação dos resultados. Entretanto, um dos pontos mais preocupantes está relacionado à confiabilidade e disponibilidade dos próprios SDIs, o que os torna vulneráveis a vários tipos de ataque [Santos and Campello, 2001].

2.3 Sistemas de Detecção de Intrusão baseados em Agentes

Agentes representam uma nova geração de sistemas de computação e constituem uma das mais recentes tecnologias de desenvolvimento de sistemas de detecção

de intrusão [Wasniowski, 2005, Hegazy et al., 2003]. Agente é um tipo de abordagem eficiente e flexível para desenvolver SDIs distribuídos.

2.3.1 Características

Segundo [Balasubramaniyan et al., 1998] existem inúmeras vantagens em se usar agentes para desenvolver sistemas de detecção, entre elas podemos citar:

- Por serem entidades autônomas, que rodam independentemente, os agentes podem ser adicionadas ou removidos do sistema sem alterar os demais componentes, conseqüentemente sem reiniciar o sistema;
- Organizando o sistema de forma hierárquica (com múltiplas camadas de agentes que reduzem dados e repassam esses dados para a camada superior) é possível tornar o sistema escalável;
- A habilidade de parar ou iniciar um agente independentemente dos outros agentes, no sistema que está sendo monitorado, adiciona a possibilidade de reconfiguração do SDI sem ter que reiniciá-lo. Se um agente com uma nova funcionalidade precisar ser incluído no sistema ou uma nova funcionalidade em um agente precisar ser inserida, o sistema como um todo não necessita ser reinicializado;
- Devido ao fato de um agente ser programado arbitrariamente, ele pode obter dados de diferentes origens (*logs*, pacotes de rede, ou outras);
- Como agentes podem ser parados e reiniciados sem afetar o restante do sistema, então podem ser atualizados para novas versões, contanto que sua interface externa não seja alterada, e outros componentes nem necessitam saber que o agente foi atualizado;
- Se um agente é implementado como um processo separado no *host*, cada agente pode ser implementado em uma linguagem de programação que melhor servir para a tarefa que ele pode executar.

2.3.2 Projetos

Abaixo são descritos, brevemente, alguns projetos de sistemas de detecção baseados em agentes:

Hegazy et al.

Hegazy et al., em [Hegazy et al., 2003], fez uso de agentes para executar a detecção de intrusão. O sistema possui quatro módulos principais: O Módulo de *Sniffing*, o Módulo de Análise, o Módulo de Decisão e o Módulo de Relatório, como se pode observar na Figura 2.1.

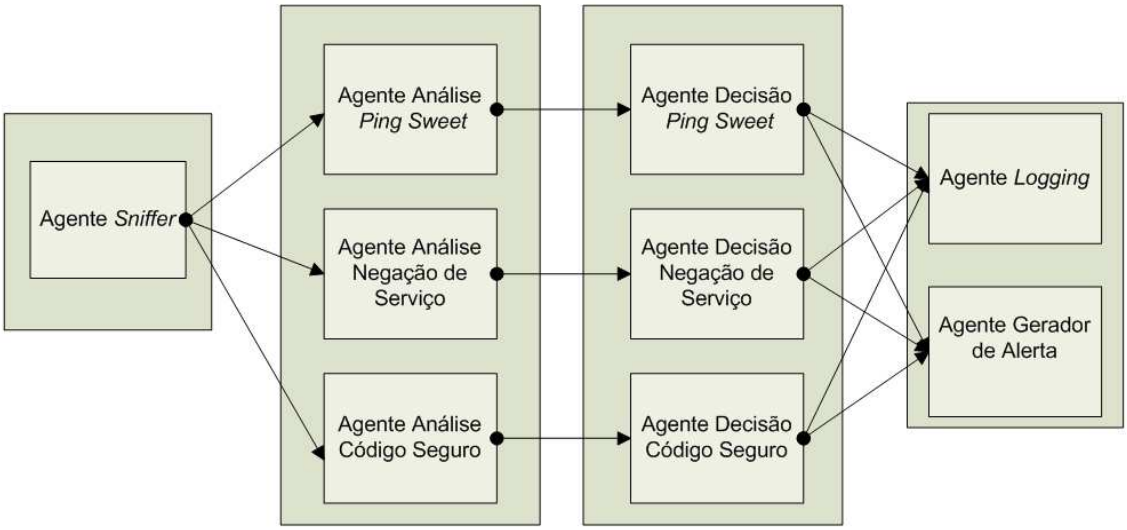

Figura 2.1: Módulos do SDI proposto por Hegazy et al.

O Módulo *Sniffing* captura pacotes da rede. O Módulo de Análise analisa os pacotes. O Módulo de Decisão toma ações de acordo com a severidade do ataque detectado. O Módulo de Relatórios gera relatórios e *logs*. O agente *sniffing* lê os pacotes da rede e os coloca em um *buffer*. O agente de análise solicita pacotes do agente *sniffing*. Ele analisa os pacotes e constrói uma lista de suspeitos de acordo com uma busca em um *buffer* de assinaturas de ataque. O agente de decisão requisita os dados e a lista de suspeitos de seu agente de análise complementar. Note na Figura 2.1 que cada tipo de ataque a ser detectado possui um agente de análise e um agente de decisão. O agente de decisão calcula o nível de severidade do ataque e a partir disso toma as ações necessárias.

Decisões e alertas são então encaminhados para o agente gerador de alertas. Agente de decisão encaminha a lista de suspeito e os dados quando é requisitado pelo agente de *logging*.

AAFID

O Projeto *Autonomous Agents For Intrusion Detection* (AAFID) usa agentes autônomos de baixo nível para coletar, analisar e filtrar dados e assim detectar comportamentos anômalos e maliciosos [Balasubramaniyan et al., 1998]. O sistema pode ser distribuído sobre um número qualquer de *hosts* em uma rede. A Figura 2.2 mostra os três componentes essenciais na arquitetura do AAFID: agentes, *transceivers* e monitores. Cada *host* pode conter um número qualquer de agentes que monitoram os eventos que estão ocorrendo naquele *host*. Todos os agentes em um *host* enviam as informações capturadas para um único *transceiver*.

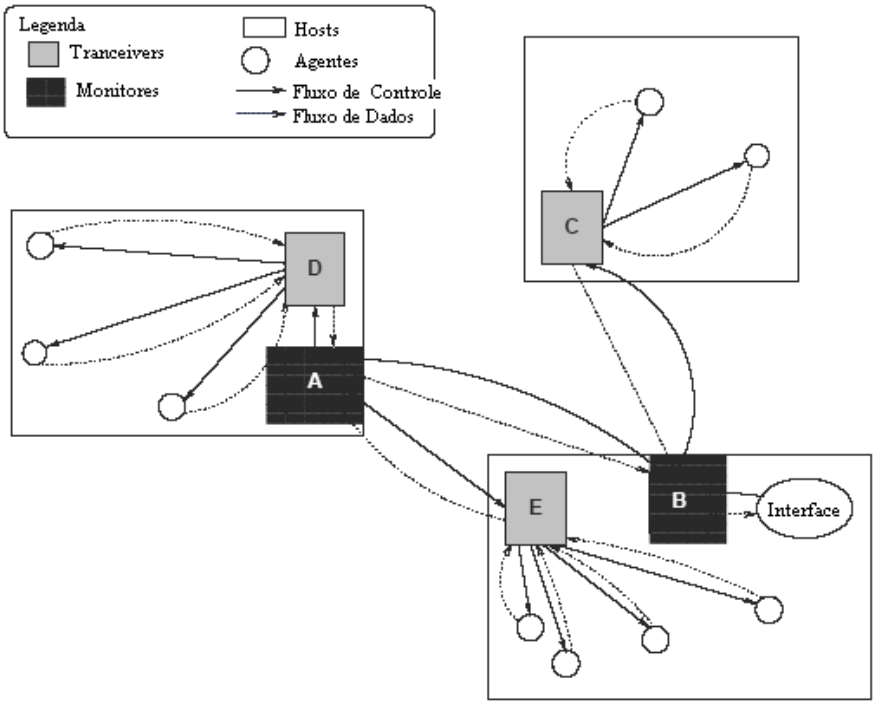

Figura 2.2: Componentes da arquitetura AAFID

Transceivers são entidades que monitoram a operação de todos os agentes em um *host*. Eles controlam os agentes em execução naquele *host* e possuem a habilidade de iniciar, parar e enviar comandos de configuração aos agentes e podem também realizar a redução dos dados recebidos dos agentes. *Transceivers* enviam relatórios para um ou mais monitores. Um único monitor pode controlar vários *transceivers* localizados em

hosts diferentes. Monitores podem ser organizados de forma hierárquica de tal forma que um monitor pode enviar informações para um monitor de mais alto nível. Também, um *transceiver* pode relatar para mais de um monitor para prover redundância e maior resistência a falha de um dos monitores. Finalmente, um monitor é responsável por prover informações e oferecer comandos de controle para uma interface de usuário.

MADIDS

Mobile Agent Distributed Intrusion Detection System (MADIDS) é um sistema baseado em agentes móveis. Este sistema é especificamente desenvolvido para processar um grande fluxo de dados transferidos em uma rede de alta velocidade [Guangchun et al., 2003]. Em MADIDS, os agentes que são configurados em cada nó processam a transferência dos dados por computação distribuída. Entretanto usando a reconfiguração dos agentes móveis, o balanceamento de carga pode ser dinamicamente executado para ganhar maior desempenho.

A Figura 2.3 mostra os principais componentes do sistema: Agente de Geração de Eventos (*Event Generation Agent*), Agente de Análise de Eventos (*Event Analysis Agent*), Agente de Localização de Eventos (*Event Tracking Agent*) e Servidor de Agente (*Agent Server*).

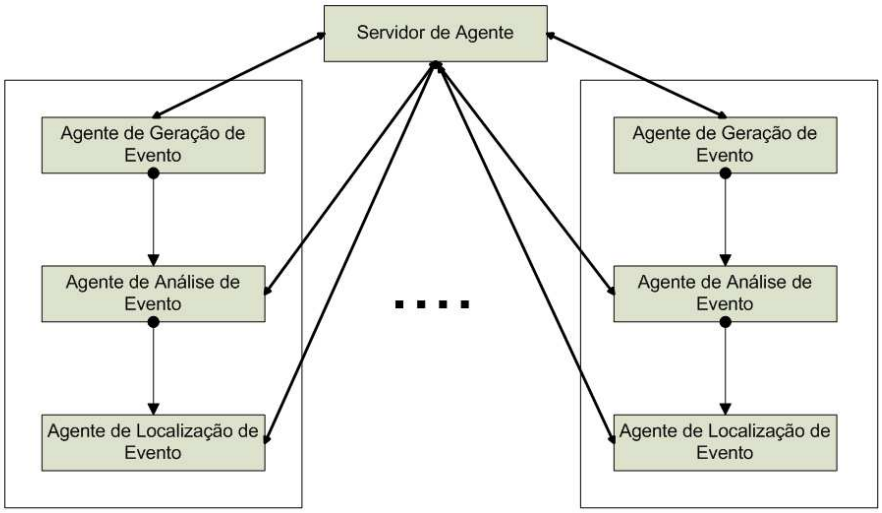

Figura 2.3: Arquitetura MADIDS

O Agente de Geração de Eventos é um sensor estacionário. Ele é responsável

pela coleta dos dados no sistema e é distribuído por toda a rede. De acordo com a carga do próprio agente e da rede, ele submete algumas partes dos dados para o seu Agente de Análise de Eventos apropriado para que este processe esses dados. Ele transfere os dados ao Servidor de Agente quando o seu Agente de Análise de Eventos não consegue gerenciá-los. Nesse caso o Agente Servidor aloca esses dados para outro Agente de Análise de Eventos processa-los. O Agente de Análise de Eventos recebe a requisição de análise e os dados, após o processamento, ele distribui o resultado do processamento para o Agente de Localização de Eventos ou Servidor de Agente de acordo com sua carga. O Agente de Localização de Eventos recebe requisições e as executa. Então ele localiza as intrusões. Quando a carga da rede permite, ele envia os dados ao Servidor de Agente. O Servidor de Agente é o supervisor central de todos os agentes. Ele possui as tarefas: alocar apropriadamente um agente ao trabalho que está sendo requisitado; monitorar e balancear dinamicamente a carga de cada agente. E ainda receber os dados do Agente de Localização de Evento e os armazenar no Servidor de Eventos.

Ramachandran e Hart

Em [Ramachandran and Hart, 2004], um sistema de detecção de intrusão baseado em agentes móveis distribuídos, que não possui um coordenador central, é apresentado. Pode ser atualizado facilmente e usado em grandes redes. A ausência de um coordenador central implica que todos os nós serão iguais. Os nós podem ser de diferentes plataformas e pertencentes a diferentes redes, fazendo assim com que o sistema seja mais robusto, de forma que uma única falha não derrube o sistema como um todo. Cada nó terá alguns vizinhos e pertencerá a uma vizinhança virtual, como mostrado na Figura 2.4. Todos os nós irão funcionar como um sistema de detecção de intrusão que irão analisar suas “casas” e a “vizinhança”.

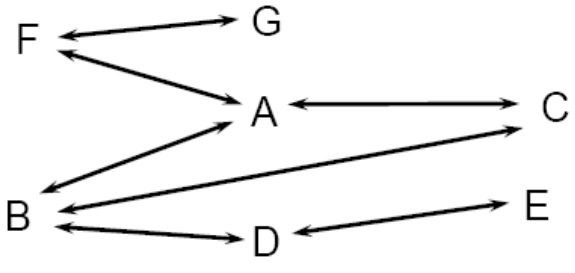

Figura 2.4: Rede de Vizinhos

Informações relacionadas a segurança de um nó serão distribuídas entre vizinhos e assim eles inspecionarão uns ao outros, periodicamente, para ter certeza que nenhuma intrusão aconteceu. Eles usarão dados distribuídos para assegurar que tudo (isto é, arquivos de dados vitais, arquivos de funções vitais) está intacto, e que não existe arquivos corrompidos e que o nó está funcionando corretamente. Quando uma intrusão é indicada, os vizinhos (do nó suspeito) são informados e uma decisão comum é feita para tomar uma ação. Este sistema usa apenas a detecção baseada em *host*.

Sua arquitetura possui os seguintes componentes: Policiais, Detetives e Chefes. Policiais são agentes móveis que possuem diferentes tipos de tarefas. Eles podem ser despachados para vários nós. Eles executam investigações nos nós para o qual são enviados e reportam os resultados de volta ao Detetive que o despachou. Os Detetives coordenam e despacham os Policiais para várias vizinhanças. Eles analisam as informações recebidas e verificam se há algo suspeito, se houver informam a suspeita ao Chefe. O Chefe, por sua vez, decide quando houve intrusão em uma vizinhança. Ele envia uma chamada de votação aos vizinhos do nó suspeito para que possa ser tomada uma ação contra o nó corrompido. A votação é baseada no voto da maioria, ou seja, metade mais um.

Zhicai et al.

Em [Zhicai et al., 2004], um modelo hierárquico de detecção de intrusão multi-nível é apresentado. O modelo utiliza as características de inteligência, mobilidade e auto-adaptabilidade dos agentes e sua capacidade de cálculo colaborativo distribuído. Assim, ele pode detectar ataques complicados efetivamente. A adoção da clonagem e da migração dos agentes e o uso de um protocolo de comunicação de segurança aumentam a segurança e a capacidade de detecção colaborativa do modelo. Dessa forma, carga de comunicação é reduzida efetivamente e intrusões podem ser detectadas e respondidas o mais rápido possível.

A adoção do modelo hierárquico é para assegurar escalabilidade. A detecção de intrusão é dividida em três níveis: rede, subrede e nó. Os SDIs correspondentes são chamados de monitor de rede, monitor de subrede e detector de nó. Eles monitoram a segurança da rede individualmente de diferentes níveis e visões. Cada monitor ou detector pode rodar autonomamente e independentemente, não interferem uns nos outros. Monitores e detectores no mesmo nível podem trocar informações assim como colaborar

para detectar ataques de rede complicados. Duas principais funções do monitor de mais alto nível é monitorar os estados dos nós associados e receber dados dos nós do próximo nível para executar correlação e detectar intrusões complicadas que envolvem diversos *hosts* e redes. A coleta dos dados é processada principalmente em nós locais e somente algumas partes são transferidas para outros nós reduzindo a carga na rede. Técnicas de detecção por assinatura e anomalia são simultaneamente utilizadas para julgar comportamentos de intrusão.

SPARTA

O sistema SPARTA [Kruegel and Toth, 2001a, Kruegel and Toth, 2001b] utiliza o paradigma de agentes móveis para a detecção de intrusão. SPARTA é um *framework* que ajuda a relacionar e a identificar eventos que podem ocorrer em diferentes *hosts* de uma rede. Cada *host* tem no mínimo um gerador de eventos locais (sensor), uma base de dados e uma plataforma de agente instalada. Os sensores locais são estáticos e funcionam de forma híbrida, podendo coletar informações tanto da rede quanto do *host*. Os eventos gerados são mantidos numa base de dados local. Para a análise dos eventos, um agente é enviado por uma console de gerenciamento com um determinado padrão de intrusão definido e uma lista de todos os *hosts* que serão visitados em uma ordem arbitrária. Um agente chegando a um *host*, procura por eventos que indiquem um padrão de ataque. Se não são localizados os padrões, o agente continua sua jornada. Quando um possível ataque é detectado, são identificadas as possíveis origens do evento e um agente auxiliar (*helper agent*) é gerado e enviado ao *host* que originou este evento para coletar mais informações. Os mecanismos de resposta a ataques são passivos e estão restritos a notificações.

A Tabela 2.1 apresenta uma análise comparativa entre os sistemas de detecção de intrusão baseados em agentes apresentados.

2.4 Tolerância a falhas

Os sistemas se tornaram mais complexos, sofisticados, mais distribuídos e operando em ambientes dinâmicos e sujeitos a falhas. Conseqüentemente, eles precisam ser mais seguros e tolerantes a falhas.

Tabela 2.1: Comparação entre SDIs Multiagentes

Projeto	Fonte de Informação	Método de Detecção	Reação	Mobilidade
Hegazy et al.	Rede	Assinatura	Ativo	Não
AAFID	Rede e *Host*	Assinatura	Passivo	Não
MADIDS	*Host*	-	-	Sim
Ramachadram	*Host*	-	Ativo	Sim
Zhicai	Rede e Host	Assinatura e Anomalia	-	Sim
SPARTA	Rede e Host	Assinatura	Passivo	Sim

A tolerância a falhas é um meio de alcançar a dependabilidade[1], trabalhando sob a suposição que um sistema contem falhas (por exemplo, falhas no desenvolvimento e na utilização dos sistemas pelos seres humanos ou ainda causada por problemas de *hardware*), e visando o fornecimento dos serviços especificados apesar da presença de falhas [Lemos and Fiadeiro, 2002].

Aplicações distribuídas com requisitos de tolerância a falhas são difíceis de serem implementadas e mantidas, principalmente se considerarmos a complexidade e as características de ambientes de larga escala. Nos sistemas distribuídos, pela natureza de distribuição do sistema, é aumentada a probabilidade de um único ponto de falha.

O principal obstáculo à implementação de tolerância a falhas é a imprevisibilidade de ocorrência de uma falha e dos seus efeitos sobre o sistema. A ação dos mecanismos de tolerância a falhas somente é iniciada com a detecção de erros já presentes no sistema, ou seja, erros resultantes de alguma falha ocorrida anteriormente. Uma vez detectado um erro, este deve ser removido e, quando necessário, o sistema deve ser reconfigurado para isolar o agente provável causador da falha. Essas atividades são chamadas, genericamente, de recuperação de erro. Após a recuperação do erro espera-se que o sistema possa prosseguir em seu funcionamento normal, sem apresentar defeito.

[1]Dependabilidade é uma propriedade vital de qualquer sistema que justifica a confiança que pode ser colocada no serviço oferecido pelo mesmo [Laprie, 1995].

2.4.1 Detecção de Erros

A detecção de erros consiste no reconhecimento da existência de um estado errôneo. Este estado errôneo consiste em um estado diferente do previsto na especificação inicial do sistema. Um esquema para prover tolerância a falhas depende de um mecanismo de detecção de erros eficiente. Um mecanismo ideal para detecção de erros deve satisfazer algumas propriedades. Primeiro, os mecanismos de detecção não devem ser afetados pelos erros existentes. Segundo, a verificação ideal deve ser completa e correta, isto implica que deve ser capaz de detectar todos os erros possíveis no comportamento do sistema e nunca detectar um erro quando ele não existe.

Com uma verificação completa e correta, pode se ter certeza de que, se um erro é detectado, existe falha. A detecção de erros num sistema pode ser feita através de mecanismos integrados ao *hardware* ou através de código adicionado ao sistema de *software*.

As técnicas básicas utilizadas por esses mecanismos são: (i) códigos redundantes, como bits de paridade, que são verificados a intervalos regulares de tempo ou sempre que a informação codificada é processada; (ii) estabelecimento de limites de tempo para que determinadas funções sejam completadas; e (iii) módulos redundantes, onde uma mesma operação é executada simultaneamente por dois componentes idênticos (réplicas) e os resultados obtidos são comparados.

2.4.2 Recuperação de Erros

A recuperação de um erro consiste em reestabelecer a consistência do estado interno do sistema. Uma vez que o erro foi detectado e sua extensão identificada, as alterações indevidas devem ser removidas, caso contrário o estado errôneo pode causar mau funcionamento no sistema futuramente. As técnicas de recuperação de erros visam transformar o estado atual incorreto do sistema em estado livre de falhas. De uma forma geral, há dois tipos de estratégias para recuperação de erros [de Castro Guerra, 2004]:

- A *Recuperação de erros por avanço (forward recovery)* tenta levar o sistema a um estado livre de erros aplicando correções ao estado danificado do sistema. Esta técnica requer um certo conhecimento dos erros que possam acontecer ao nível da aplicação. Exceções e tratamento de exceções constituem um mecanismo comumente

aplicado para a provisão de recuperação de erros por avanço;

- A *Recuperação de erros por retrocesso (backward recovery)* tenta retornar o sistema para um estado prévio livre de erros, não requerendo nenhum conhecimento específico dos erros que possam ocorrer. Como falhas de *software* e os erros causados por elas são de natureza imprevisível, recuperação de erros por retrocesso é geralmente aplicada para tolerar esse tipo de falha. Bancos de dados transacionais e mecanismos de *checkpoint/restart* são exemplos de sistemas que empregam recuperação de erros por retrocesso. Quando as características das falhas são bem conhecidas, recuperação de erros por avanço proporcionam uma solução mais eficiente.

2.4.3 Redundância

Redundância é um modo efetivo para alcançar tolerância a falhas em sistemas distribuídos [Marin et al., 2001, Guessoum et al., 2002]. O uso da redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- Redundância de *hardware*: replicação de *hardware* está baseada na replicação de componentes. Ela é eficiente e cara, e é a técnica mais utilizada em aplicações críticas.

- Redundância de *software*: a simples replicação de *software* não é uma solução adequada. Portanto outras formas de replicação são utilizadas: diversidade (ou programação n-versões); blocos de recuperação; verificação de consistência, entre outras técnicas.

- Redundância de informação: na replicação de informação, bits ou sinais extras são armazenados ou transmitidos junto ao dado, sem que contenham qualquer informação útil. Exemplos: códigos de paridade, *checksums*, duplicação de código e códigos cíclicos.

- Redundância de tempo: este tipo de replicação repete a computação no tempo. Evita custo de *hardware* adicional, mas aumenta o tempo necessário para realizar uma computação. É usada em sistemas onde tempo não é crítico, ou onde o processador trabalha com ociosidade.

Todas essas formas de redundância: de *hardware*, de *software*, temporal e de informação, tem algum impacto no sistema, seja no custo, no desempenho, na área (tamanho, peso) ou na potência consumida. Assim, o uso da redundância em qualquer projeto deve ser bem ponderada.

Um componente de *software* replicado é definido como um componente de *software* que possui uma representação em dois ou mais *hosts* [Guerraoui and Schiper, 1997]. Réplicas idênticas de um mesmo componente (por exemplo, distribuídos) aumentam a disponibilidade e a confiabilidade do sistema, porém implica em um aumento da complexidade do sistema com custo maior de comunicação e processamento. As principais estratégias de replicação utilizadas são replicação ativa e replicação passiva.

Na *replicação ativa* todas as réplicas processam todas as mensagens de entrada concorrentemente. Conseqüentemente, esta estratégia consome mais memória e tempo de processador e pode conduzir a um alto *overhead*, dependendo do tamanho do sistema. Porém, provê um tempo de recuperação pequeno [Marin et al., 2001].

Na *replicação passiva* só uma única réplica processa a mensagem de entrada e transmite seu estado atual periodicamente às outras réplicas a fim de manter consistência. Com isto, economiza utilização de processador porque só ativa uma réplica redundante em caso de falha. Esta técnica requer menos recursos de CPU do que a replicação ativa, mas envolve um gerenciamento de *checkpoint* que pode se tornar caro em tempo de processamento e espaço [Marin et al., 2001].

A escolha da estratégia mais adequada é diretamente dependente do contexto do ambiente, especialmente a taxa de falhas e as exigências da aplicação em termos de tempo de recuperação e *overhead*. A estratégia ativa deve ser escolhida se a taxa de falha se tornar muito alta ou ainda se o projeto da aplicação especificar duras restrições de tempo. Em todos os outros casos, a estratégia passiva é preferível segundo [Marin et al., 2001].

2.4.4 Replicação Adaptativa

Mecanismos de replicação têm sido aplicados em sistemas distribuídos para prover tolerância a falhas, no entanto, essa replicação tem sido feita na maioria dos casos de forma estática, ou seja, definida pelo programador e aplicada antes da aplicação iniciar [Guessoum et al., 2002].

No entanto, modelos adaptativos têm se mostrado bastante adequados para serem aplicados quando o objetivo é manter requisitos de qualidade de serviço (QoS). Se o requisito é a tolerância a falhas, o uso de técnicas adaptativas pode também se mostrar interessantes em sistemas onde as condições de carga e a própria evolução das aplicações não são de todo previsíveis.

Com adaptabilidade, é possível manter o nível da confiabilidade do sistema em um padrão aceitável. O uso das técnicas de adaptabilidade permite, por exemplo, que em uma replicação de software se possa adicionar ou remover réplicas de um serviço de acordo com as variações do seu ambiente de execução, alcançando assim a otimização no uso dos recursos do sistema. No entanto, para se perceber essas variações no ambiente, tais como carga no processamento, comunicação, entre outras, se fazem necessários mecanismos para obtenção dessas informações, ou seja, é necessário desenvolver formas adequadas de monitoramento do ambiente.

No caso de aplicações multiagentes dinâmicas e adaptativas a importância do agente pode evoluir dinamicamente no decorrer da computação. Além disso, a disponibilidade dos recursos é limitada. Conseqüentemente, replicação simultânea de todos os agentes de um sistema de larga escala não é possível [Guessoum et al., 2002]. Contudo, a replicação de agentes específicos, que são identificados como cruciais para uma determinada aplicação, pode permitir evitar este problema facilmente [Marin et al., 2001, Guessoum et al., 2005].

2.5 Tolerância a Falhas em Sistemas Multiagentes

A principal motivação para usar sistemas multiagentes é a natureza distribuída das informações, recursos e ações [Marin et al., 2001]. A natureza modular do sistema multiagente dá um certo nível de tolerância a falhas inerente, porém a natureza não determinística dos agentes, o ambiente dinâmico e a falta de um ponto de controle central torna impossível prever estados falhos e torna o gerenciamento do comportamento de falhas imprevisível. A menor falha de um único agente pode se propagar ao longo de todo o sistema e pode conduzir o sistema inteiro a falhar [Fedoruk and Deters, 2002].

A área de tolerância a falhas foi melhorada através de décadas de pesquisa. Tolerância a falhas tem sido pesquisada em varias áreas da computação, tais como,

arquitetura de computadores, sistemas operacionais, sistemas distribuído, computação móvel e rede de computadores. Apesar desses avanços, cada nova área tem seus conjuntos de desafios para o qual as técnicas passadas tem aplicabilidade limitada [Chetan et al., 2005].

De acordo com [Zhang, 2005], há uma necessidade crescente por aperfeiçoar a tolerância a falhas em sistemas multiagentes. Já que, segundo [Marin et al., 2001], a maioria das aplicações e plataformas multiagentes distribuídas atuais ainda não trata, de forma sistemática, essa possibilidade de falhas.

Muitos trabalho tem sido feitos em detecção de falhas e tolerância a falhas em diversas áreas [Saidane et al., 2003], [Liang et al., 2003], [Townend and Xu, 2003], [Santos and Campello, 2001], [Chetan et al., 2005], [Zorzo and Meneguzzi, 2005]. Também, diversas abordagens para tolerância a falhas em sistemas multiagentes são apresentadas na literatura [Zhang, 2005], [Klein and Dellarocas, 1999]. Dentre elas algumas são mostradas a seguir.

2.5.1 Framework DarX

DarX (Dynamic Agent Replication eXtension) é um *framework* para projetar aplicações distribuídas confiáveis que incluem um conjunto de agentes distribuídos [Guessoum et al., 2002, Marin et al., 2001, Marin et al., 2003]. Cada agente pode ser replicado em um número não limitado de vezes e com estratégias diferentes de replicação (ativa e passiva). DarX inclui gerenciamento de grupo para dinamicamente adicionar e remover réplicas. Também provê *multicast* atômico e ordenado para comunicação interna dos grupos de réplicas. O número de réplicas e a estratégia interna de um grupo específico de agentes são totalmente escondidos dos outros agentes da aplicação. Cada grupo possui exatamente um líder que se comunica com os outros agentes. O líder também checa a vitalidade de cada réplica e é responsável pelo *broadcasting* confiável. Em caso de falha do líder, uma nova réplica é eleita líder entre as réplicas restantes no grupo. Como não é possível sempre replicar todos os agentes do sistema porque os recursos são usualmente limitados, a idéia é dinamicamente e automaticamente aplicar mecanismos de replicação *onde* (agentes) e *quando* é mais necessário de acordo com a importância do agente para o sistema em diferentes momentos do processamento. Somente os agentes críticos são replicados. O problema seria identificar a importância dos agentes para escolher a melhor

estratégia de replicação e o número adequado de réplicas. A solução foi encontrada usando as seguintes abordagens: informações de nível semântico e informações ao nível de sistema [Guessoum et al., 2005]. DarX tem a vantagem de dinamicamente adaptar o número de réplicas para reduzir a complexidade de sistema.

O *framework* DarX usa uma estratégia de recuperação *backward* uma vez que o agente inicia do último estado difundido. Porém, se o estado do último agente não for um estado livre de erro, então o agente replicado também falhará (fracasso). Conseqüentemente, o *framework* DarX não provê um mecanismo de controle para assegurar que a réplica executará corretamente. Além disso, o *framework* DarX provê uma estratégia para reduzir o número de agentes a ser reproduzido.

2.5.2 Sistemas Multiagentes *Brokered*

Sistemas multiagentes freqüentemente necessitam de *brokers* para aceitar requisições, redirecionar requisições e respostas, compartilhamento de informações, gerenciar o sistema e para várias outras tarefas [Kumar et al., 1999]. Porém, estes sistemas são propensos a fracassos do *broker*. Na realidade, um sistema multiagente que depende de *brokers*, pode ficar indisponível se um ou mais *brokers* do sistema se tornarem inacessíveis devido a fracassos como *crash* de máquina, avaria na comunicação, morte do processo *broker*, fracassos de processos e numerosas outras falhas de *hardware* e *software*. Em [Kumar et al., 1999, Kumar and Cohen, 2000], uma técnica de recuperação de falha de *broker* é apresentada. A técnica é baseada em replicação de *brokers*. Ela é aplicada quando houver vários agentes *brokers* em um sistema multiagente. Os *brokers* são organizados em grupos hierárquicos que são usados para comunicação e coordenação entre os agentes. Essa técnica concentra a tolerância a falhas no time de *brokers* e não nos agentes individualmente. Estes agentes *brokers* podem ser capazes de substituir qualquer agente *broker* que fica indisponível. Conseqüentemente, o sistema multiagente pode continuar operando contanto que haja pelo menos um agente *broker* que permanece no time de *brokers*. Os agentes que estavam comunicando com o *broker* que falhou, subscreverão com um novo *broker* e reiniciarão a comunicação deles. Essa técnica requer computação extra para a administração de camadas de *brokers*.

Esta técnica usa uma abordagem de recuperação *forward*. O sistema é movido de um estado falho a um estado livre de erro. O sistema não continua operando do último

estado livre de erro. No caso de fracassos, agentes reiniciam as comunicações deles. Ela não descreve o número de *brokers* a serem replicados nem o número de réplicas por *brokers*.

2.5.3 Sistemas Multiagentes usando *Proxies*

A. Fedoruk e R. Deters, em [Fedoruk and Deters, 2002], propõem o uso de *proxies* para tornar transparente o uso de replicação de agentes, isto é, permitir que réplicas de um agente se comportem como uma entidade única diante dos outros agentes. A abordagem de replicação transparente minimiza a complexidade e carga do sistema. Um *proxy* nada mais é do que uma entidade computacional que provê interface para um conjunto de réplicas de agentes. O *proxy* gerencia o estado das réplicas e todas as comunicações internas e externas do grupo são redirecionadas para ele. Contudo, isso aumenta a carga de trabalho do *proxy* que é uma entidade central e ele pode se tornar um gargalo para o sistema. Para torná-lo confiável, uma hierarquia de *proxies* para cada grupo de réplicas deve ser construída. A replicação é feita pelo programador antes da execução. Essa abordagem é cara porque trabalha com grupo de réplicas de todos os agentes do sistema.

Esta técnica usa uma abordagem de recuperação *backward*. O sistema é movido de um estado falho a um estado livre de erro. Uma pilha de estados é utilizada para recuperar o estado de um agente falho.

2.5.4 Sentinelas

Haegg, em [Haegg, 1997], propôs o uso de sentinelas para monitoramento de sistemas. Uma sentinela é um agente e sua missão é proteger funções específicas ou proteger contra estados específicos na sociedade de agentes. A sentinela não participa na solução do problema, mas pode intervir se necessário, escolhendo métodos alternativos para resolver o problema, excluindo agentes falhos, alterando parâmetros para agentes e relatando ao operador humano. Sendo um agente, a sentinela interage com outros agentes usando semântica de endereçamento. Assim ele pode, pelo monitoramento da comunicação dos agentes e pela interação (perguntando) construir modelos de outros agentes. Ele pode usar temporizadores para detectar falhas de agentes ou de um elo de comunicação falho. Dado um conjunto de agentes que cooperam em realizar funções do

sistema, uma sentinela é configurado pra guardar aquela função. As sentinelas mantêm modelos dos agentes. Alguns itens em tal modelo são diretamente copiados do modelo do mundo dos agentes em interesse tornando-se partes do modelo das sentinelas. Esses itens são denominados *checkpoints* e permitem as sentinelas julgar o estado de um agente, não somente de seu comportamento, mas de seu estado interno. Isso é uma forma de detecção prematura de agentes falhos e da inconsistência entre os agentes, que é considerada uma falha do sistema. Adicionar sentinelas no sistema parece ser uma boa abordagem, porém as próprias sentinelas representam pontos de falhas para o sistema multiagente.

2.5.5 Agentes Móveis

Agentes móveis podem viajar de um servidor a outro para procurar informação ou executar tarefas nos servidores visitados. Um sistema multiagente, composto de agentes móveis, pode ser propenso a dois tipos de fracassos [Mishra, 2001]: fracassos de servidor e fracassos de agente. Nós apresentamos, nas próximas subseções, técnicas que foram propostas para lidar com ambos os tipos de fracassos.

SG-ARP: uma abordagem para recuperação de servidor

O Grupo de Servidor baseado na abordagem SG-ARP *(Server Group based Agent Recovery Protocol)* é descrito em [Mishra, 2001]. Ele permite que os agentes móveis executem corretamente apesar de fracassos de servidor. Para superar um fracasso de servidor, cada servidor é reproduzido várias vezes. O servidor e suas réplicas definem um grupo de servidor. Os membros de um grupo de servidor dividem entre eles a carga trazida pelos agentes visitantes. Os diferentes servidores compartilham uma área de armazenamento na qual eles armazenam os seus estados. Quando um servidor tiver falhado, todos os agentes que estavam executando no servidor que falhou são distribuídos aos membros restantes do grupo. Uma vez que o sistema usa a abordagem *backward*, poderia haver informações perdidas quando os agentes fossem distribuídos aos servidores restantes.

Esta abordagem usa recuperação *backward*, uma vez que servidores compartilham os estados deles. Quando um servidor falhar, o sistema volta a um estado livre de erro de forma que agentes podem executar em outros servidores existentes. Porém, não é especificada quantas vezes os servidores são reproduzidos.

Abordagem de Recuperação de Falhas de Agentes

Na literatura, várias abordagens são propostas para recuperar de fracassos de agente em sistemas de agente móveis [Serugendo and Romanovsky, 2002]. Um agente pode ser confrontado a três fontes de fracassos:

1. O fracasso do componente no qual o agente está executando;
2. Fracasso de outros agentes com que o agente está cooperando;
3. Fracasso do próprio agente.

Um das abordagens propostas [Serugendo and Romanovsky, 2002] é a abordagem de Meta-agente: cada agente é associado com um meta-agente que é responsável pelo aspecto tolerante a falhas. O meta-agente permite o agente a controlar exceções. Nesta abordagem, o meta-agente precisa de outro meta-agente uma vez que também é propenso a fracasso. Conseqüentemente, tolerância a falhas não está garantida.

2.5.6 Grade Computacional

Outro domínio no qual técnicas de tolerância a falhas podem ser aplicadas é a grade computacional. Grades computacionais são ambientes de computação com recursos volumosos tais como servidores para processamento e armazenamento de dados. Em [Townend and Xu, 2003], um esquema de tolerância a falhas baseado em replicação é proposto. Essa abordagem tem a vantagem de não requerer *software* adicional no cliente e tira vantagem do grande número de recursos disponíveis na grade. Os recursos da grade são registrados num repositório. Os recursos da grade possuem um serviço de tolerância a falhas que é capaz de receber tarefas, executá-las, realizar operações de *checksum*, e enviar o resultado de volta ao cliente. A aplicação cliente solicita a execução da tarefa para um serviço de coordenação. Este serviço pesquisa em um ou mais repositórios de serviços quais são os locais e números de recursos compatíveis e disponíveis. Esse serviço de coordenação também é replicado. Quando um cliente envia uma tarefa, o serviço de coordenação determina a quais "nós" devem ser enviadas réplicas da tarefa e então faz o envio. Após cada nó completar a tarefa, o serviço de tolerância a falhas realiza um *checksum* baseado no resultado que o nó gerou. Posteriormente, ele envia o resultado ao serviço de coordenação. Após um dado número de nós terminar a execução, o

serviço de coordenação vota no *checksum* retornado ((número de nós/2)+ 1), e seleciona aleatoriamente um nó com *checksum* correto para retornar o resultado correto para o cliente. O serviço de coordenação envia o *checksum* ao cliente que o compara com o resultado recebido. Uma desvantagem óbvia é o aumento do *overhead*.

2.6 Conclusões

Neste capítulo, alguns conceitos relacionados a sistemas multiagentes, sistemas de detecção de intrusão e tolerância a falhas em sistemas multiagentes foram apresentados.

Em sistemas multiagentes, a definição de agente, suas características e classificação foram abordadas. Na seção de sistemas de detecção de intrusão, as características, a classificação e as limitações desse tipo de sistema foram apresentadas. Na seção de projetos de sistemas de detecção baseados em agentes, alguns projetos e suas principais características foram apresentados. E por fim, uma comparação entre as características dos projetos foi apresentada.

Pudemos observar nos projetos de sistemas de detecção de intrusão, mostrados neste capítulo, que nenhuma referência a tolerância a falhas é citada. Até mesmo os projetos mais atuais ainda não estão aplicando soluções para esse problema.

Em tolerância a falhas, replicação e replicação adaptativa foram descritas mostrando as vantagens e desvantagens de cada estratégia.

Finalmente, o uso de tolerância a falhas em alguns projetos de sistemas multiagentes foi apresentado. Pode-se constatar que a maioria dos projetos utiliza a redundância de algum componente para prover a tolerância a falhas.

3 O Projeto NIDIA

Este capítulo aborda o Projeto NIDIA. As principais características do projeto são apresentadas. A arquitetura inicial e a atual, e todos os agentes que as compõem são mostrados. A funcionalidade de cada agente também é explicada neste capítulo.

3.1 Características

O Projeto ***N****etwork* ***I****ntrusion* ***D****etection System based on* ***I****ntelligent* ***A****gents* (NIDIA), proposto por [Lima, 2002] e que está sendo desenvolvido na Universidade Federal do Maranhão, é um sistema de detecção de intrusão baseado no conceito de sociedade de agentes inteligentes. O sistema é capaz de detectar novos ataques através de uma rede neural, em tempo real.

No monitoramento uma combinação de agentes sensores, que são instalados em pontos estratégicos da rede (analisando pacotes de tráfego) e em *hosts* críticos (analisando *logs*), é adotada. Estes agentes possuem o objetivo de capturarem pacotes suspeitos e atividades maliciosas e a partir disso gerar um índice de suspeita de ataques [Santos and Nascimento, 2003, Dias and Nascimento, 2003]. Esta abordagem adota um modelo híbrido que mesclasse ambas as formas de coleta de dados elevando o nível de segurança do sistema [Lima, 2002].

O NIDIA também provê a metodologia de detecção por abuso e anomalia, garantindo uma robustez maior ao sistema, e possui a capacidade de interagir com sistemas tipo *firewall* [de Lourdes Ferreira, 2003, Oliveira et al., 2005], no intuito de diminuir os problemas apresentados. Essa integração permite que seja alcançado um maior nível de segurança, uma vez que os dois sistemas possuem características complementares.

A escolha da arquitetura multiagente para o sistema foi devido a necessidade de se obter as seguintes vantagens [Dias, 2003, Pestana, 2005, Hegazy et al., 2003]:

- Facilidade de manutenção e atualização do sistema com adição e remoção de agentes, mantendo o máximo de disponibilidade do mesmo;

- Possibilidade de atualização dos agentes responsáveis pelo mecanismo de identificação de ataques;
- Capacidade de se obter maior adequação a determinado ambiente, através da utilização de agentes especializados para um computador ou rede em particular;
- Maior tolerância a falhas do que sistemas monolíticos que possuem um único ponto crítico de falhas;
- Alto potencial de escalabilidade através da adição de novos agentes para manter a performance exigida em sistemas em expansão;

3.2 Arquitetura Inicial

Segundo [Dias, 2003] a arquitetura inicial do projeto NIDIA foi inspirada no modelo lógico do *Common Intrusion Detection Framework* (CIDF). O CIDF [CIDF, 2005, Pestana, 2005] é um projeto da *Defense Advanced Research Projects Agency* (DARPA), cujo objetivo é desenvolver protocolos e interfaces de programação, de forma que projetos de pesquisa em detecção de intrusão possam compartilhar informações e recursos. E também, permitir que diferentes componentes de um sistema de detecção de intrusão possam compartilhar informações da forma mais detalhada e completa. E ainda, que um sistema possa ser reutilizado em contextos diferentes do qual ele foi originalmente configurado [IDMEF, 2006].

O projeto NIDIA possui agentes com função de geradores de eventos (agentes sensores), mecanismos de análise dos dados (agentes de monitoramento e de avaliação de segurança), mecanismos de armazenamento de histórico e um módulo para realização de contramedidas (agente controlador de ações) [Pestana, 2005], obedecendo a estrutura definida pelo CIDF. Além disso, existem também agentes responsáveis pela integridade do sistema (Agente de Integridade do Sistema), pela coordenação das atividades do SDI como um todo (Agente de Segurança Local) e pela atualização das bases de conhecimento (Agente de Atualização do Sistema) [Lima, 2002].

Na Figura 3.1 é mostrada a arquitetura do NIDIA proposta por [Lima, 2002].

Segue abaixo um descritivo das funcionalidades dos agentes e demais elementos

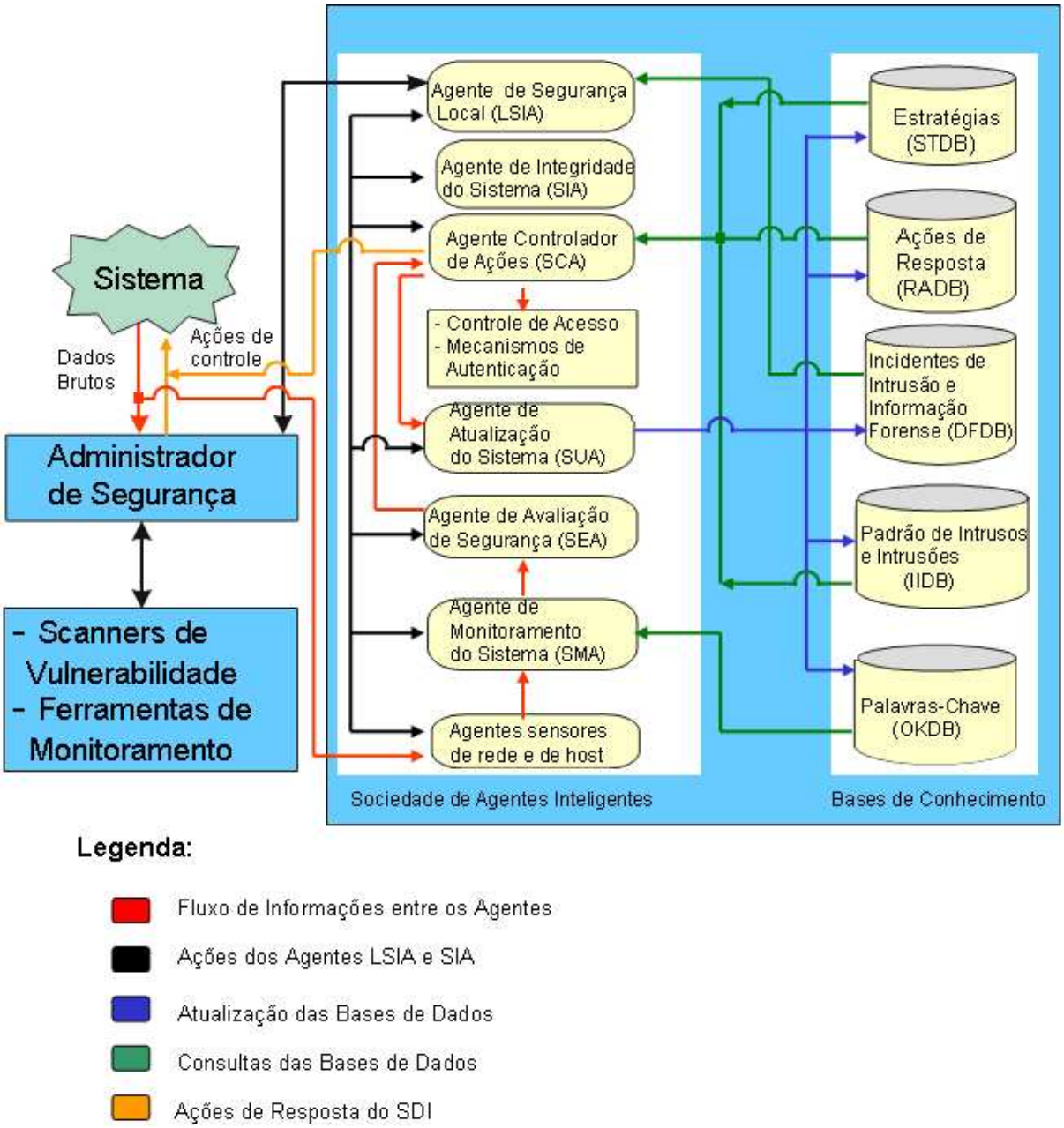

Figura 3.1: Arquitetura NIDIA - Inicial

da arquitetura inicial:

- *Bases de conhecimento*: o NIDIA possui bases de dados com função de armazenar as estratégias para a sua política de segurança (STDB), as contramedidas a serem tomadas em caso de atividade suspeita (RADB) e os padrões de ataque utilizados para a detecção (IIDB). Além disso, possui uma base de dados responsável por armazenar os incidentes de intrusão (DFDB), que podem servir de fundamento para investigar e provar a culpa de um determinado atacante.

- *Agentes Sensores*: estes agentes funcionam como os "sentidos receptores" do

sistema, com a função de capturar o que está ocorrendo no meio exterior. Ele possui duas categorias: agentes sensores de rede (capturando pacotes) e agentes sensores de *host* (coletando informações em *logs* de servidores).

- *Agente de Monitoramento de Sistema (SMA)*: este agente recebe os dados dos agentes sensores, realiza uma pré-formatação e repassa para o agente de avaliação de segurança.

- *Agente de Avaliação de Segurança do Sistema (SEA)*: agente responsável por emitir um grau de suspeita sobre os eventos que foram previamente formatados. Para isso, utiliza bases de conhecimento, como a base de dados de intrusos e intrusões (IIDB), a base de dados de incidentes de intrusão e informação *forense* (DFDB) e a base de estratégias (STDB).

- *Agente de Atualização do Sistema (SUA)*: agente responsável pela atualização das bases de conhecimento.

- *Agente Controlador de Ações (SCA)*: com base no parecer do SEA, este agente deve tomar uma contramedida de acordo com as bases de dados de estratégia (STDB) e de ações (RADB).

- *Agente de Integridade do Sistema (SIA)*: agente responsável por manter o sistema resistente à subversão, evitando que agentes do próprio SDI possam fazer parte de algum esquema de ataque.

- *Agente de Segurança Local (LSIA)*: este agente funciona como o gerente de toda a sociedade de agentes e também como interface com o administrador do sistema.

- *Agente Administrador do Sistema*: é o próprio administrador do sistema (agente humano) que realiza as atribuições de configurar as bases de dados do sistema, ativar/desativar agentes e analisar os alertas dados pelo sistema a respeito da cadeia de dados coletados.

O funcionamento do NIDIA, a partir do agente sensor de rede, é explicado a seguir, resumidamente, com o auxílio do diagrama de colaboração mostrado na Figura 3.2, em notação UML (*Unified Modeling Language*) [UML, 2005].

Inicialmente os agentes sensores de rede capturam pacotes do tráfego da rede e realizam duas tarefas:

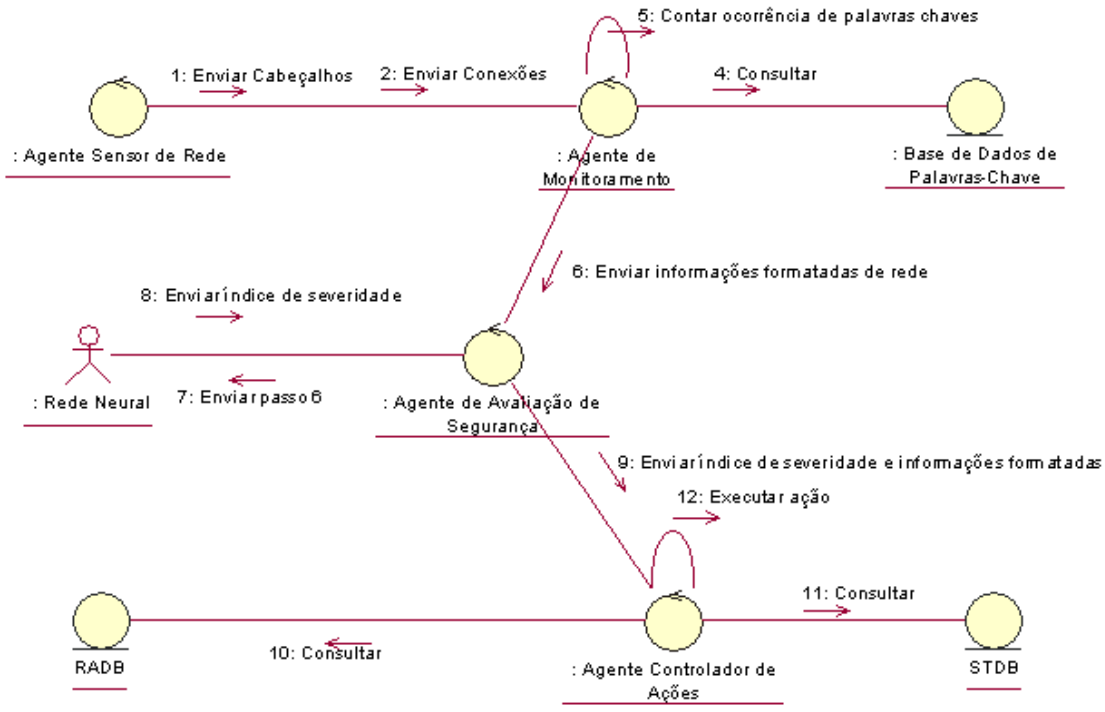

Figura 3.2: Funcionamento NIDIA (Arquitetura Inicial)

1. Enviam o cabeçalho dos pacotes para os Agentes de Monitoramento do Sistema (SMA) de rede específicos para formatar esse tipo de informação e;

2. Constroem as conexões TCP (*Transmission Control Protocol*) entre servidor-cliente a partir do conteúdo dos pacotes recolhidos e as enviam para os agentes SMA de rede específicos para formatar esse tipo de informação.

Os agentes SMA realizam a tarefa de formatar os dados recebidos. No caso específico do tratamento das conexões TCP montadas pelo agente sensor de rede, o agente SMA especializado consulta um banco de dados de palavras-chaves e realiza a contagem de ocorrência das mesmas, tendo como produto final um vetor de contagem de palavras-chave para cada conexão.

Posteriormente, os agentes SMA enviam esses vetores para que os Agentes de Avaliação de Segurança (SEA) possam fazer uma análise deles. Existem agentes SEA especializados em tratar o cabeçalho dos pacotes através de filtros e agentes SEA especializados em inspecionar o conteúdo das conexões TCP.

O SEA repassa os dados formatados para uma rede neural em Java e tem como produto final um índice de severidade para uma determinada conexão TCP, cujo valor

encontra-se na faixa de 0 (zero) a 1 (um), representando atividades normal e suspeita, respectivamente.

Esse índice de severidade é enviado para o Agente Controlador de Ações (SCA) que é o agente responsável pelo controle das ações de resposta que o sistema deve tomar em caso de uma tentativa de invasão.

De posse do índice de severidade recebido do SEA, o SCA identifica o ataque e consulta as bases de dados de estratégia (STDB) e ações de resposta (RADB) tendo como objetivo responder à invasão.

3.3 Arquitetura Atual

Devido à evolução natural do projeto e a necessidade de adição de novos agentes, uma nova arquitetura foi introduzida por [Siqueira and Abdelouahab, 2006]. Essa arquitetura é baseada em camadas, que estão de acordo com a funcionalidade dos agentes. Essa arquitetura é mostrada na Figura 3.3 e suas camadas são descritas a seguir, assim como as responsabilidades de cada agente pertencente a cada camada.

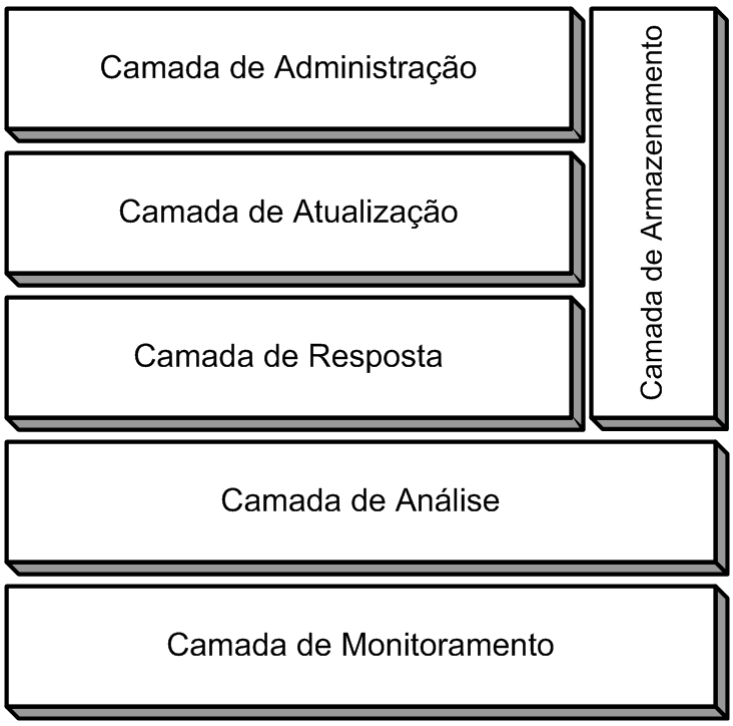

Figura 3.3: Arquitetura em camadas do NIDIA (Arquitetura Atual)

3.3.1 Camada de Monitoramento

Camada responsável por capturar eventos e fornecer informações sobre os mesmos para o restante do sistema. Nesta camada estão localizados os agentes sensores que se dividem em dois tipos:

- *Agente Sensor de Rede*: captura os pacotes que estão trafegando na rede. Ele interage com a rede de forma passiva, somente escutando o seu tráfego como um *sniffer*[1] e tentando interferir o mínimo possível no desempenho da rede e ainda tentando não corromper o tráfego. Após a captura do pacote estes agentes realizam duas tarefas:

  1. enviam o cabeçalho do pacote para o agente de monitoramento do sistema (SMA) e;
  2. constroem conexões TCP entre cliente e servidor a partir do conteúdo dos pacotes recolhidos e envia para o SMA.

- *Agente Sensor de Host*: trabalha coletando informações em *logs*, em tempo real, de um *host* em particular e disponibilizando-as para análise. Baseado em suas responsabilidades, este agente pode ser subdivido em:

  i. *Log*: interage com o mecanismo de auditoria do sistema operacional e os *logs* do *host*;

  ii. Alerta: agente responsável apenas por informar quando um determinado *host* (*honeypot* ou servidor de produção) foi alcançado. Seus alertas são encaminhados à camada de reação;

  iii. Coleta: coleta no *host* informações sobre um usuário específico.

3.3.2 Camada de Análise

Camada responsável pela análise dos eventos recebidos da camada de monitoramento. Nessa camada, os eventos coletados são formatados permitindo que padrões de ataque e/ou comportamentos anormais na rede e nos servidores monitorados sejam identificados.

[1]Um *sniffer* é um programa que consegue capturar todo o tráfego que passa em um segmento de uma rede.

- *Agente de Monitoramento do Sistema (SMA)*: organiza e realiza uma pré-formatação nos eventos coletados pelos agentes sensores. O SMA se divide em três tipos:

  i. *Host*: é responsável pela formatação somente das informações coletadas pelo agente sensor de *host*;

  ii. Rede-Cabeçalho: formata o cabeçalho do pacote enviado pelo sensor de rede;

  iii. Rede-Conteúdo: formata o conteúdo do pacote enviado pelo sensor de rede através de uma consulta a uma base de dados otimizada de palavras-chaves (OKDB).

- *Agente de Avaliação de Segurança do Sistema (SEA)*: é responsável por emitir um grau de suspeita sobre os eventos que foram previamente formatados. Para auxiliar nessa tarefa, ele faz uso de informações da Base de Dados de Incidentes de Intrusão e Informação Forense (DFDB) e da Base de Dados de Estratégias (STDB). A saída desse agente é um nível de alerta associado à severidade do evento. Este se divide em três tipos:

  i. *Host*: avalia as informações formatadas pelo SMA de *Host*;

  ii. Rede-Cabeçalho: trata o cabeçalho dos pacotes através de filtros, que podem ser baseado em regras;

  iii. Rede-Conteúdo: analisa o grau de suspeita de um ataque através de uma rede neural MLP (Perceptrons de Múltiplas Camadas) previamente treinada. Gera um grau de severidade, entre 0 e 1, e quanto mais próximo de 1 maior a suspeita de ataque.

- *Agente BAM (BA)*: é responsável pela identificação do nome do ataque através de um vetor de contagem de palavras chaves e do grau de severidade que recebe do SEA. Após a análise repassa o resultado ao Agente Controlador Principal (MCA).

3.3.3 Camada de Reação

Camada responsável pelo controle das ações que o sistema deve tomar em caso de uma tentativa de invasão. No projeto inicial esta camada era representada apenas como um único agente [Lima, 2002]. No entanto, tornou-se complexa agora sendo representada pelos seguintes agentes [de Lourdes Ferreira, 2003, Oliveira, 2005]:

- *Agente Controlador Principal (MCA)*: recebe o nome do ataque do BA, consulta mais informações sobre o ataque e sobre o atacante. A seguir, ele as envia ao Agente de Avaliação de Severidade. Após receber o nível de segurança, consulta a Base de Dados de Ações de Respostas (RADB) e a Base de Dados de Estratégias (STDB) para iniciar a reação apropriada.

- *Agente de Avaliação de Severidade*: mede o nível de perigo que uma intrusão está oferecendo para o sistema. A partir do índice de severidade repassado pelo MCA, este agente irá gerar um nível *fuzzy* de segurança indicando o estado do sistema (normal, alerta ou emergência). Esses níveis representam o grau de severidade do ataque.

- *Agente de Proteção de Host*: é responsável por reconfigurar *hosts* para um estado de segurança intensificado. Esse agente pode ser instruído a mudar o estado de disponibilidade de portas e serviços, bloqueando assim o acesso, ou desligar o *host* (medida extrema utilizada quando um sistema de arquivos precisa ser preservado para casos de perícia *forense*).

- *Agente de Reconfiguração Dinâmica da Rede*: é responsável por reconfigurar *firewalls* e *routers*, de modo a limitar o acesso do intruso aos *hosts*.

- *Agente de Desvio de Tráfego*: agente responsável por desviar o tráfego malicioso para uma unidade de decepção com serviços similares aos do servidor de produção que for alvo de um ataque.

- *Agente de Traceroute Remoto*: agente responsável por tentar descobrir o percurso até o atacante. O objetivo de fornecer informações sobre a origem verdadeira do tráfego, mesmo que o endereço fonte seja falso, de modo que o administrador possa reprimir ou limitar a ação do atacante.

3.3.4 Camada de Atualização

Camada responsável pela atualização das bases de dados. As consultas são feitas por agentes de qualquer camada, porém inserções e atualizações são feitas somente através desta camada. Ela possui também a responsabilidade de manter a integridade e consistência das informações armazenadas. Nela estão inseridos os seguintes agentes:

- *Agente Monitor* : envia constantemente requisições às bases de dados de arquivos de ataques em busca de novos itens de informação. Quando ele recebe a informação de que um novo arquivo está disponível, ele o recupera e o envia ao Agente Gerador de Conexões.

- *Agente Gerador de Conexões*: cria arquivos texto com o conteúdo das conexões de ataques, assim como um arquivo de índice correspondente. Da mesma forma arquivos para as conexões normais e o arquivo de índice serão criados. Após a criação destes arquivos, o Agente Gerador de Conexões informa ao Agente Surrogate sobre a disponibilidade destes novos arquivos.

- *Agente Surrogate*: cria representações internas dos arquivos de conexões, atualiza a base de dados de assinaturas de ataques com essas representações e informa ao Agente Construtor de SAARA (Sociedade Atualizada de Agentes de Reconhecimento de Ataques) que novas representações internas foram produzidas. Como existe o arquivo de índice que relaciona uma conexão de rede a um determinado ataque, a base de dados de assinaturas de ataques será constituída pelo nome do ataque e a representação interna do arquivo espelho para a respectiva conexão.

- *Agente Construtor de SAARA*: o agente Construtor de SAARA tem a responsabilidade de construir novas SAARAs de acordo com a metodologia descrita em [Dias, 2003]. Nessa metodologia vetores de contagem de palavras-chave, tanto de conexões suspeitas quanto de conexões normais, são utilizados para o treinamento supervisionado de uma rede neural MLP que será responsável por emitir alertas indicando o grau de suspeita de determinada conexão.

- *Agente de Interface*: agente do tipo reativo que serve de intermediário entre a Agência Central de Segurança e o sistema. Ele tem como função receber requisições dos SDIs multiagente e informá-los sobre a existência de uma nova SAARA.

- *Agente Analisador de XML*: é responsável por analisar os alertas de segurança recebidos pelo Agente Monitor e atualizar a Base de Dados de Ações de Respostas (RADB) com as informações desses alertas.

3.3.5 Camada de Administração

Camada responsável pela administração e integridade de todos agentes do sistema.

- *Agente de Integridade do Sistema (SIA)*: responsável por garantir a integridade do sistema, evitando que agentes do próprio sistema possam estar fazendo parte de algum ataque. Ele busca por eventos inesperados ou que diferem do perfil normal dos agentes ativos. Conseqüentemente, ele interage com todos os agentes do sistema.

- *Agente de Segurança Local (LSIA)*: responsável pelo gerenciamento dos agentes que compõem o sistema e pela interface entre o sistema e o administrador de segurança. Através dele é possível gerenciar o estado e a configuração dos agentes, a atualização da base de dados de estratégias (STDB), o registro das ocorrências detectadas e das ações tomadas pelo sistema como resposta a estas ocorrências. É o único agente que se comunica diretamente com o administrador do segurança.

3.3.6 Camada de Armazenamento

Camada responsável por armazenar informações relevantes à detecção de intrusão. O sistema possui sete bases de dados citadas em [Lima, 2002], [Pestana, 2005] e [Oliveira, 2005]:

- *Base de Dados de Incidentes de Intrusão e Informação Forense (DFDB)*: armazena os danos causados por ataques bem sucedidos, tentativas de ataques e ações de resposta que foram tomadas pelo sistema.

- *Base de Dados de Intrusos e Intrusões (IIDB)*: contém as assinaturas de intrusão que serão utilizadas para a detecção de atividades suspeitas. Ela deve ser constantemente atualizada para garantir que novas técnicas de ataque possam ser detectadas.

- *Base de Dados de Palavras-Chave (OKDB)*: armazena as palavras-chave que serão utilizadas pelo agente SMA para formatar as conexões recebidas do agente sensor de rede.

- *Base de Dados de Ações de Respostas (RADB)*: contém as informações referentes às ações de resposta que devem ser tomadas de acordo com a severidade de um ataque

detectado. Varia de acordo com a política de segurança de cada organização.

- *Base de Dados de Estratégias (STDB)*: nessa base, as estratégias adotadas pela organização em relação a sua política de segurança são registradas. Permite a adaptabilidade do sistema aos mais diversos tipos de organizações.

- *Lista Negra*: serve para registrar resumidamente os eventos sobre as fontes potencialmente hostis que o sistema capturar através dos agentes de alerta que estarão distribuídos através dos *honeypots* [Oliveira, 2005].

- *Lista Branca*: nessa lista são registrados todos os eventos envolvendo as fontes que acessaram os recursos de produção do ambiente. Esses dados podem ser úteis para eventuais consultas e análises no caso de comprometimento comprovado de algum *host*.

- *Log de Incidentes Reportados*: *log* onde as ações executadas em um *host* são registradas.

3.4 Comparação entre SDIs Multiagentes e o NIDIA

A Tabela 4.1 apresenta uma análise comparativa entre os sistemas de detecção de intrusão baseados em agentes apresentados no capítulo 2 e o NIDIA.

Tabela 3.1: Comparação entre SDIs Multiagentes e o NIDIA

Projeto	Fonte de Informação	Método de Detecção	Reação	Mobilidade
Hegazy et al.	Rede	Assinatura	Ativo	Não
AAFID	Rede e *Host*	Assinatura	Passivo	Não
MADIDS	*Host*	-	-	Sim
Ramachadram	*Host*	-	Ativo	Sim
Zhicai	Rede e Host	Assinatura e Anomalia	-	Sim
SPARTA	Rede e Host	Assinatura	Passivo	Sim
NIDIA	Rede e Host	Assinatura e Anomalia	Ativo	Sim

3.5 Conclusões

Neste capítulo o Projeto NIDIA, suas características e suas arquiteturas (inicial e atual) foram apresentados. A arquitetura inicial do projeto e a arquitetura atual, proposta por [Siqueira and Abdelouahab, 2006] foram apresentadas. Da atual arquitetura a funcionalidade de cada camada e dos agentes pertencentes a cada uma delas foi descrita.

De acordo com [Vuong and Fu, 2001] a falta de mecanismos de tolerância a falhas em sistemas de agentes inteligentes restringe severamente o escopo de sua aplicabilidade. E ainda, o comportamento autônomo do agente e a natureza maliciosa da Internet elevam a importância da segurança, tanto do agente quanto do seu ambiente de execução. Com isso fica claro que, para a implementação de um sistema de detecção de intrusão baseado em agentes suficientemente robusto, é importante ter mecanismos de segurança que devem, obrigatoriamente, cooperar na busca por um sistema mais confiável, seja do ponto de vista funcional (maior segurança) como da tolerância a falhas nos próprios agentes. No próximo capítulo, uma mecanismo de tolerância a falhas para o NIDIA é proposto.

4 Mecanismo de Tolerância a Falhas

O objetivo deste capítulo é mostrar o mecanismo de tolerância a falhas proposto ao NIDIA. Primeiramente, os objetivos e requisitos do mecanismo são apresentados. Em seguida, a arquitetura do mecanismo e seus componentes são detalhados. Finalmente, as detecções de falhas abordadas pelo mecanismo são apresentadas.

4.1 Objetivos

Segundo [Campello et al., 2001] "[...] é evidente, principalmente em ambientes distribuídos, a preocupação em manter a confiabilidade de todos os módulos existentes no sistema, garantindo, no mínimo, um comportamento livre de falhas (*fail-safe*)[...]"

Falhas são inevitáveis, mas as suas conseqüências como a parada do sistema, a interrupção no fornecimento do serviço, a perda de dados e a perda de comunicação podem ser evitadas pelo uso adequado de mecanismos de tolerância a falhas, que devem ser preferencialmente, viáveis e de fácil compreensão.

O mecanismo de tolerância a falhas proposto tem como objetivo detectar falhas no NIDIA e prover uma adequada recuperação para os diferentes tipos de falhas detectadas. Ou seja, garantir que o NIDIA continue oferecendo seus serviços, mesmo que falhas ocorram.

O mecanismo utiliza duas abordagens: o monitoramento do sistema (*hosts* e agentes) [Haegg, 1997] e a replicação de agentes [Fedoruk and Deters, 2002, Guessoum et al., 2005] para prover a tolerância a falhas. Ele tem os seguintes objetivos no NIDIA:

a) Detectar agentes inativos;

b) Detectar agentes maliciosos;

c) Detectar a mudança de prioridade (importância) dos agentes no decorrer do

processamento do sistema e, assim, ter a capacidade de alterar a estratégia de replicação de cada agente.

As falhas devem ser detectadas através de diagnóstico preciso e completo e a recuperação deve ser apropriada para cada tipo de falha. Ambos, diagnóstico e recuperação devem ser providenciados no menor intervalo de tempo possível e com o mínimo de impacto ao NIDIA.

É importante ressaltar que o mecanismo aborda apenas falhas que podem ocorrer nos agentes do NIDIA. Outro tipos de falhas, tais como, parada (*crash*) de um *host*, problemas na comunicação não são abordados nesse trabalho.

4.2 Requisitos

Mecanismos de tolerância a falhas, em sua maioria, exigem componentes adicionais ou algoritmos especiais, o que os tornam caros devido o aumento na complexidade do sistema onde são adotados. Para a escolha adequada de um mecanismo de tolerância a falhas, as características especiais da aplicação e as suas exigências quanto a confiabilidade e a disponibilidade devem ser conhecidas em detalhes. A confiabilidade (do inglês *reliability*) é a capacidade que um sistema tem em se manter funcionando, mesmo que ocorram falhas. A disponibilidade (do inglês *availability*) é a probabilidade de um sistema estar funcionando em um dado instante.

Alguns requisitos foram definidos para o mecanismo de tolerância a falhas proposto. Eles são descritos a seguir:

- Mudanças mínimas: o sistema original deve ser modificado o mínimo possível, ou seja, os agentes do NIDIA devem sofrer o mínimo de alteração na implementação dos seus comportamentos originais;
- O mecanismo deve ser, ele próprio, tolerante a falhas;
- Escalabilidade: o mecanismo deve ser flexível e escalável para suportar potencial crescimento do sistema em tamanho e complexidade. Deve ser capaz de suportar o aumento no número de agentes do NIDIA, tanto com relação a redundância como adição de novos agentes com novas funcionalidades;

- Não intrusivo: o mecanismo deve ser o menos intrusivo possível, afim de não comprometer o funcionamento do NIDIA.

Um aspecto importante para o mecanismo de tolerância a falhas proposto é a comunicação confiável entre os agentes. A confiança da entrega das mensagens trocadas entre os agentes do NIDIA é feita pela utilização de um MTP (*Message Transport Protocol*) que conta com a implementação de uma especificação XML (*eXtensible Markup Language*) que fornece padrões de comunicação confiável. Esta confiança foi desenvolvida por [Oliveira and Abdelouahab, 2006]). Agentes utilizam um MTP para se comunicarem com outros agentes. Um MTP trata-se de uma implementação de um protocolo de comunicação para ser utilizado no transporte das mensagens. A especificação que o MTP implementa é a WS-RM (*WS-Reliable Messaging*). A WS-RM fornece um protocolo que destinatários e remetentes utilizam para garantir que uma mensagem enviada foi realmente recebida através de um modelo de envio de confirmações. Resumidamente, sempre que uma mensagem é recebida uma confirmação é enviada para o remetente.

Portanto, sempre que for mencionado comunicação entre agentes neste trabalho, estamos nos referindo a comunicação confiável feita através do MTP que implementa a especificação WS-RM, acima mencionados.

4.3 Arquitetura

O mecanismo de tolerância a falhas foi introduzido no NIDIA através de um agente denominado Agente de Tolerância a Falhas (*System Fault Tolerance Agent* - SFTA) e uma base de dados denominada Base de Dados de Perfis (*Profile Database* - PRDB). O Agente de Tolerância a Falhas pertence à camada de administração, já que uma das responsabilidades dessa camada é manter a integridade do sistema.

O Agente de Tolerância a Falhas está desenvolvido como uma sociedade de agentes. Esses agentes cooperam entre si e executam suas tarefas de forma independente. O objetivo da sociedade é dividir as tarefas, tornando os agentes menos complexos e tornando a atualização mais flexível. A Figura 4.1 mostra os componentes da arquitetura do mecanismo.

A sociedade de agentes é composta por três agentes: Agente Sentinela (*System Sentinel Agent*- SSA), Agente de Avaliação de Falhas (*System Fault Evaluation Agent* -

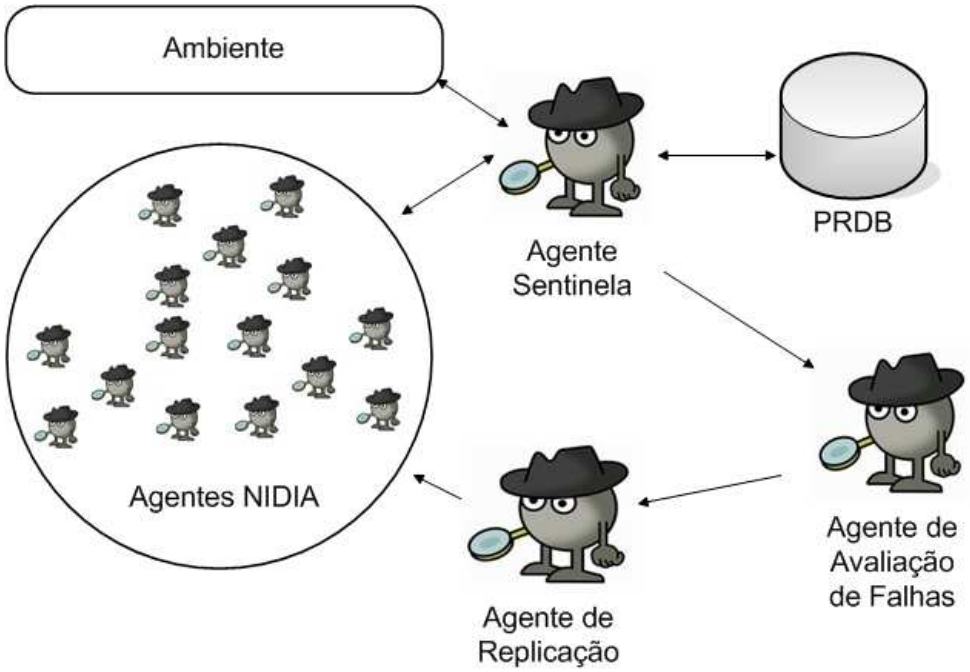

Figura 4.1: Arquitetura do Mecanismo de Tolerância a Falhas

SFEA) e Agente de Replicação (*System Replication Agent* - SRA). A Base de Dados de Perfis é utilizada pelos agentes para auxiliá-los na realização de suas tarefas.

Os Agentes Sentinelas estão distribuídos pelos *hosts* que possuem agentes do NIDIA. Ou seja, em cada *host* que existir, pelo menos, um agente do NIDIA em execução, deverá haver também um Agente Sentinela. Ele é responsável pelo monitoramento dos agentes naquele *host* e pelo monitoramento do próprio *host*. Esse monitoramento é realizado através da coleta de informações sobre CPU, memória e disco (nos *hosts*) e mensagens (nos agentes). E quando, a partir desse monitoramento, uma falha for detectada deverá ocorrer uma ação de recuperação adequada para cada tipo de falha. Essa decisão é tomada pelo Agente de Avaliação de Falhas. O Agente de Replicação é responsável pelo gerenciamento da replicação dos agentes.

As características e responsabilidades de cada componente do mecanismo são mostradas, com detalhes, a seguir.

4.3.1 Agente Sentinela (SSA)

O Agente Sentinela é o principal agente no mecanismo de tolerância a falhas. Esse agente é responsável pela detecção de falhas no sistema. Essa detecção é feita através do monitoramento do sistema. Esse monitoramento permite descobrir se agentes estão

ativos ou inativos.

O monitoramento é utilizado também para inferir a necessidade de mudança de estratégia de replicação dos agentes e necessidade de migração de agentes entre *hosts*.

Um monitoramento nas ações dos agentes do NIDIA permite determinar se estas são autorizadas ou não. A execução de uma ação não autorizada caracteriza a presença de um agente malicioso no sistema. Um agente malicioso é aquele que executa e/ou requisita a execução de uma ação não autorizada e essa ação pode corromper tanto *hosts* quanto agentes.

O monitoramento é realizado através da coleta de informações nos agentes e nos *hosts*. As informações coletadas no agente são descritas a seguir:

- *Quantidade de mensagens recebidas*: essa informação é utilizada como parâmetro para determinar a importância do agente para o sistema num momento do processamento. Uma mensagem recebida representa para um agente uma ação que deverá ser executada. Portanto, podemos concluir que a quantidade de mensagens recebidas representa a quantidade de ações que o agente deverá executar. Consequentemente, quanto mais mensagens são recebidas num determinado intervalo de tempo mais importante é o agente para o sistema naquele momento;

- *Mensagens recebidas*: baseado nas mensagens recebidas é possível determinar se o comportamento de um agente não está correto. A troca de mensagens entre agentes é predefinida e, portanto, quando um agente recebe uma mensagem de um agente que não está autorizado a fazer esse envio, isso significa que existe um agente malicioso no sistema.

As informações coletadas no *host* (disponibilidade do processador, da memória e do disco) são utilizadas pelo mecanismo de tolerância a falhas para inferir as decisões de mudança de estratégia de replicação dos agentes, migração de agentes e o balanceamento de carga entre os agentes. Baseados nela, é possível mensurar a disponibilidade *host* para *hospedar* novos agentes.

Os Agentes Sentinelas estão distribuídos, ou seja, estão executando nos *hosts* que possuem pelo menos um agente do NIDIA executando nele. Eles estão organizados de forma hierárquica, a fim de prover maior escalabilidade e também tolerância a falhas. A Figura 4.2 mostra como estão estruturados esses agentes.

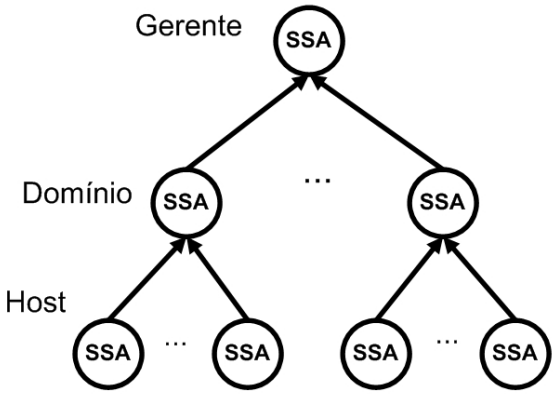

Figura 4.2: Hierárquia do Agente Sentinela

O SSA *Host* é responsável por monitorar os agentes que executam no *host* em que ele reside. E, portanto, existirá tantos SSAs *Hosts* quanto existirem *hosts* com agentes do NIDIA. O SSA Domínio é responsável por monitorar os SSAs *Hosts*. Caso um SSA *Host* falhe outro do mesmo tipo deverá substituí-lo para executar a tarefa de monitoramento. O SSA Gerente é responsável pelo monitoramento dos SSAs Domínio. Em caso de falha do SSA Gerente, o mesmo procedimento de recuperação deve ser realizado, ou seja, um novo agente deve ser iniciado para realizar sua tarefa. Essa hierarquia pode ser reduzida, ou seja, podemos ter apenas o nível de SSA *Host* e SSA Domínio. Essa estrutura vai depender da distruibuição do NIDIA pela organização. O SSA Gerente deve ser introduzido caso seja preciso distruibuir o NIDIA por várias redes. No caso de termos apenas uma rede, teremos um SSA *Host* para cada *host* e um SSA Domain para monitorar esses SSA *Hosts*.

Utilizando essa hierarquia, o sistema pode ser recuperado e continuar a monitorar as atividades dos agentes, caso ocorra uma falha de *crash* em um Agente Sentinela.

4.3.2 Agente de Avaliação de Falhas (SFEA)

Esse agente é responsável pela análise das informações coletadas pelos SSAs. Ele tem uma visão global do sistema, uma vez que recebe informações relacionadas a todos os *hosts* e agentes. Isso facilita o processo de decisão e, conseqüentemente, permite mensurar:

- A necessidade de mudança de estratégia de um agente. Uma vez que sua importância no sistema foi alterada e sua estratégia de replicação não está de acordo com sua

realidade. Um agente que é muito solicitado em um intervalo de tempo é considerado muito importante para o sistema aquele momento, portanto deve ser elevada a preocupação com a falha desse agente. E portanto, sua estratégia de replicação deve estar de acordo com essa situação. No caso de um agente estar sendo muito pouco solicitado, ou não ter nenhuma solicitação para ele nesse intervalo de tempo, sua estratégia de replicação deve ser a mais barata (replicação ativa) ou até mesmo não será necessário replicá-lo;

- Que ação de recuperação deve ser executada para cada tipo de falha. Ele pode requisitar: a alteração da estratégia de replicação de um grupo de agentes, migração, criação de um novo agente ou exclusão de um agente considerado malicioso;

- A disponibilidade de recursos nos *hosts*. Podendo assim detectar quando um *host* está sobrecarregado ou próximo desse estado ou quando ainda possui disponibilidade de recursos para agregar mais agentes. Baseado nessas informações, este agente pode solicitar a migração de agentes para que um balanceamento de carga entre os *hosts* possa ser realizado.

4.3.3 Agente de Replicação (SRA)

O Agente de Replicação é responsável pela organização do grupo de réplicas de agentes: adicionando ou removendo réplicas. Ele conhece a estratégia de replicação de cada grupo e, portanto, pode alterá-la quando for necessário. Também coordena a consistência das réplicas no grupo.

A replicação utilizada é transparente para os agentes do NIDIA. Somente o Agente de Replicação conhece os grupos, as estratégias de replicação, a quantidade de réplicas de cada grupo e a localização das réplicas.

Cada agente do NIDIA possuirá um Agente de Replicação que é encarregado de cuidar do grupo de réplicas.

4.3.4 Base de Dados de Perfil (PRDB)

A Base de Dados de Perfil (PRDB) é a base de dados que armazena informações utilizadas pelo mecanismo de tolerância a falhas. Ela é um repositório de metadados e

dados.

Um repositório de metadados permite completa representação de um metamodelo (pacotes, classes, atributos, associações, etc) dentro do repositório, disponibilizando formas de criar, acessar e redefinir objetos pertencentes a ele [Lopes, 2006].

Ela armazena informações que representam o perfil de cada agente, tal como, acesso a arquivo e bases de dados. Ela guarda informações sobre o modelo de comportamento do NIDIA, como o fluxo de mensagens entre os agentes. E também informações sobre grupo de réplicas de agentes. Resumindo, ela armazena todas as informações que serão utilizadas pelos agentes do mecanismo para que possam realizar suas tarefas.

4.4 Replicação de agentes

A replicação de agentes é uma abordagem bastante utilizada para prover tolerância a falhas em sistemas multiagentes [Fedoruk and Deters, 2002, Guessoum et al., 2005]. A idéia básica da replicação de agentes é que quando uma falha de comunicação ou *host* resulta na falha de uma réplica de um agente, outra réplica daquele agente continuará executando e completando a tarefa do agente.

No entanto, os recursos disponíveis num sistema são limitados. Então a replicação simultânea de todos os agentes de um sistema grande não é possível. A solução para tal problema seria então dinamicamente e automaticamente aplicar mecanismos de replicação.

O mecanismo proposto utiliza a replicação de agentes e é projetado a fim de adaptar a estratégia de replicação de acordo com a importância do agente no decorrer do processamento. Conseqüentemente, todos os agentes não têm a mesma estratégia de replicação em um dado momento da computação. As estratégias de replicação adotadas são: replicação ativa e replicação passiva, ambas descritas no capítulo 2. Com isso, alguns agentes estarão sendo replicados com a estratégia de replicação passiva, outros com a replicação ativa e outros nem mesmo precisarão ser replicados em certos momentos.

A introdução de réplicas conduz ao aumento na complexidade e carga no sistema. Porém, a vantagem da abordagem de replicação adaptativa é a redução de custo

no desenvolvimento do mecanismo dentro do sistema uma vez que a replicação somente será aplicada quando (no momento mais apropriado) e onde (agente) for necessário. Podendo acontecer que uma grande maioria de agentes não precise ser replicada ou que use a replicação passiva, que possui um custo menor que a replicação ativa.

4.4.1 Grupo de Agentes

O mecanismo utiliza o conceito de *Grupo de Agentes* (*Agent Group* - AG) que consiste no conjunto de todas as réplicas que correspondem ao mesmo agente. Sempre que o NIDIA iniciar um novo agente, será criado um AG contendo uma única réplica.

Durante o decorrer da computação o número de réplicas dentro de um AG pode variar, mas deverá conter no mínimo uma réplica trabalhando ativamente para assegurar que a computação que foi requerida para aquele agente será de fato processada. Assim, no caso de falhas, se no mínimo uma réplica está funcionando, o agente correspondente não esta perdido para a aplicação.

Cada grupo terá uma única estratégia de replicação.

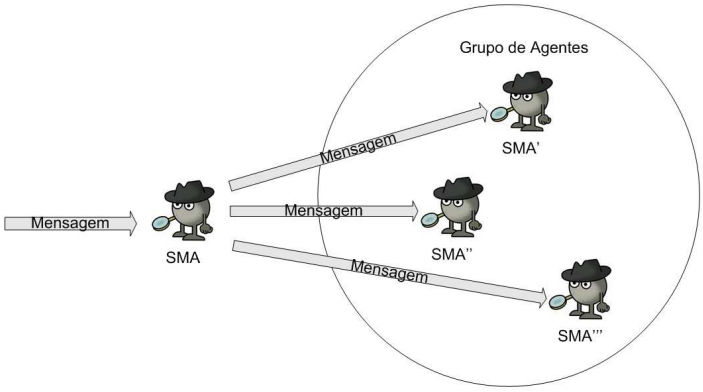

Figura 4.3: Grupo de Agentes

4.5 Detecção de Falhas

Um aspecto fundamental para obter tolerância a falhas é a detecção de erros. Isso requer o conhecimento do estado atual do sistema e do estado correto desejado. O erro é uma discrepância entre esses dois estados.

O mecanismo, além da replicação, utiliza a abordagem de monitoramento para prover a tolerância a falhas. O monitoramento permite desencadear reações adequadas, por exemplo, executar ações de emergência, equilibrar a carga de trabalho do sistema e assegurar adequados graus de tolerância a falhas.

O monitoramento permite ao mecanismo detectar diferentes tipos de falhas. *Crash* de agentes, sobrecarga e *crash* de *hosts* e problemas na rede de comunicação são algumas das falhas que podem ocorrer em sistemas multiagentes. Agentes podem morrer (parar) devido a condições inesperadas, gerenciamento impróprio de exceções e outros *bugs*.

Três tipos de detecção, através do monitoramento, são feitas pelo mecanismo: detecção de agente malicioso, detecção de *crash* de agente e detecção da mudança de importância do agente para prover mudança de estratégia de replicação dos agentes.

4.5.1 Detecção de Agente Malicioso

Um ataque pode corromper o código e o estado interno do agente e, conseqüentemente, modificar seu comportamento original. Um agente pode ser corrompido enquanto está executando em um determinado *host* ou ser alterado por outro agente malicioso, o que resulta em uma mudança do seu comportamento original. Segundo [Vuong and Fu, 2001], a corrupção de um agente pode ser feita através da adição de algumas ações maliciosas ou pela remoção de comportamentos existentes originalmente. Para resolver esse problema, o mecanismo propõe um método para detectar quando o comportamento do agente está corrompido.

Esse método de detecção é realizado utilizando uma lista de capacidades associada ao monitoramento dos agentes. A *Lista de Capacidades* relaciona a identidade de um agente com seus privilégios dentro do sistema. Privilégios correspondem a acessos autorizados a determinados recursos. O recurso pode ser um arquivo, uma base de dados ou até mesmo a comunicação com um outro agente. Com a lista de capacidades, o mecanismo de tolerância pode especificar e controlar até que ponto um agente com certa identidade está usando adequadamente os recursos, ou seja, executando adequadamente o comportamento pré-definido para ele.

O processo de monitoramento dos agentes realizado pelo mecanismo utiliza

duas abordagens: o auto-monitoramento e o monitoramento externo.

O *auto-monitoramento* é aquele que é feito internamente pelo próprio agente. A tarefa de checar suas próprias ações é acrescida às funcionalidades dos agentes dentro do sistema. A desvantagem dessa abordagem é que ela confia na avaliação do agente. E este estando corrompido não fará uma avaliação correta. Os agentes, porém são responsáveis por avaliar somente suas ações.

A Figura 4.4 mostra o diagrama de estados genérico dos agentes. Nele podemos observar como é feita o auto-monitoramento.

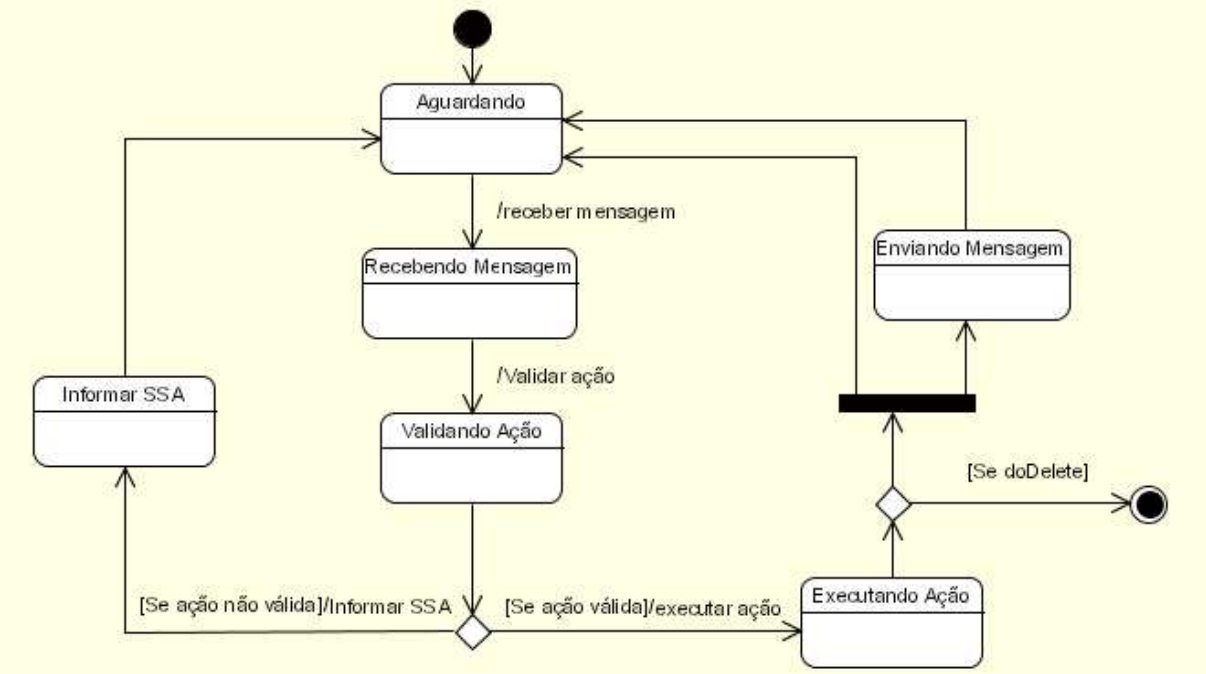

Figura 4.4: Diagrama de Estados - Agentes NIDIA

O agente ao ser ativado fica aguardando pelo recebimento de mensagens. Quando uma mensagem é recebida, o agente irá validar a execução da ação solicitada. Essa validação é feita verificando se essa ação está incluída na sua lista de capacidades, ou seja, se o agente tem autorização para executá-la.

Se a ação for válida, o agente a executa. Após a execução, o agente pode ir ao estado de aguardando, ou seja, pronto para receber uma nova mensagem. Ele pode também enviar uma mensagem de resposta ao agente que solicitou a ação, ou encaminhar uma solicitação a outro agente e, em seguida, passar para o estado de aguardando.

Se ação for inválida, ele deve encaminhar uma mensagem de alerta ao Agente Sentinela. E em seguida, ir para o estado de aguardando.

Caso ocorra alguma exceção no agente, o método *DoDelete()* é chamado e o agente é exlucído (morto) do sistema.

No entanto, por não confiar totalmente na validação do agente também é adotado o monitoramento externo. O *monitoramento externo* ocorre quando a avaliação do comportamento (perfil) é externa ao agente. Nessa abordagem, um agente que não participa do funcionamento normal do sistema, ou seja, não coopera para o objetivo final do NIDIA possui a função de monitorá-lo. Esse monitoramento externo é função desempenhada pelo Agente Sentinela.

O monitoramento externo é realizado pela observação das ações executadas pelos agentes. Baseado nessas ações é possível detectar se existe algum agente agindo de forma anormal. A desvantagem dessa abordagem é que o comportamento errôneo deve ocorrer primeiramente para depois ser detectado. Porém possui a vantagem de separar o controle do sistema das funcionalidades do mesmo.

O monitoramento externo dos agentes segue os seguintes passos:

1. observar o agente,
2. checar se o perfil do agente é consistente com a observação e finalmente,
3. restaurar o comportamento normal do sistema.

É necessário representar as ações autorizadas como um modelo de comportamento do agente e comparar esse modelo com a ação que está sendo executada. Em [Wallace, 2005], que utiliza a mesma abordagem de monitoramento é utilizado um modelo de restrições.

A Figura 4.5 mostra o diagrama de seqüência da detecção de uma agente malicioso, usando a notação UML [UML, 2005]. Essa detecção é realizada através do monitoramento externo dos agentes. Os passos dessa detecção são descritos a seguir:

i. O Agente Sentinela consulta o agente para verificar que ações estão sendo executadas. Para cada ação executada haverá uma consulta à PRDB;

ii. Ele checa se a ação é autorizada para o agente que a está executando;

iii. Se ação é autorizada, continua o processo de monitoramento;

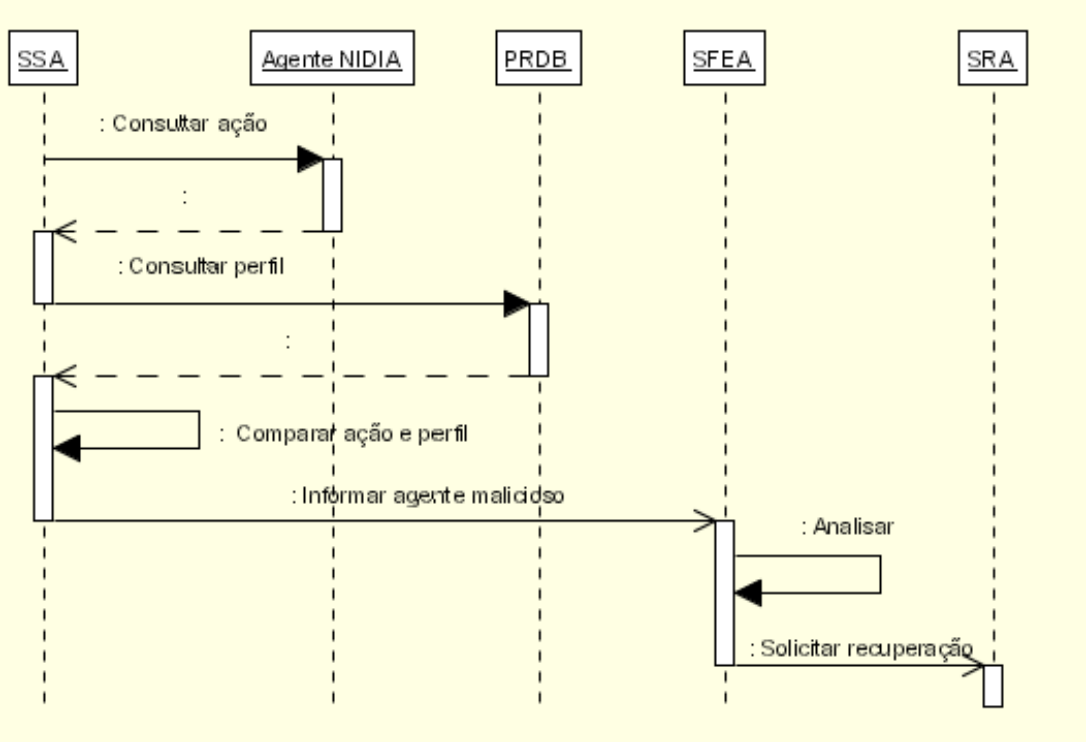

Figura 4.5: Diagrama de Seqüência - Detecção de Agente Malicisoso

iv. Se ação não é autorizada uma mensagem de alerta é enviada ao Agente de Avaliação de Falhas;

v. O Agente de Avaliação de Falhas verifica na sua base de regras quais as ações que são executadas para este tipo de falha;

vi. O Agente de Avaliação de Falhas após analise informa ao Agente de Replicação que um agente malicioso deve ser desativado e uma outra réplica desse agente deve ser ativada.

Um agente somente pode receber solicitação e executar uma ação se o agente que solicitou estiver na sua lista de agentes autorizados. Sempre que um agente malicioso é descoberto, ele deve ser isolado dos demais agentes, ou seja, toda a comunicação com esse agente deve ser interrompida. Este agente deve ser eliminado do sistema e todas as mensagens enviadas a ele e que não foram ainda processadas devem ser repassadas a outro agente do mesmo tipo.

4.5.2 Detecção de *Crash* de Agentes

O *crash* de um agente representa uma falha muito grave em um sistema de agentes cooperativos. Uma vez que eles estão empenhados em realizar tarefas menores que compõem uma tarefa complexa, se algum agente "parar" o objetivo final do sistema não será alcançado. Esse tipo de falha pode conduzir a parada total do sistema.

No NIDIA, os agentes são cooperativos e cada agente possui uma funcionalidade específica. No caso de ocorrer a parada de determinados agentes, o funcionamento do sistema como um todo é comprometido. Um exemplo crítico desse tipo de falha seria a parada do Agente Sensor de Rede. Se este agente tiver seu funcionamento interrompido, o sistema todo fica interrompido.

O mecanismo de tolerância a falhas, através do Agente Sentinela, realiza um monitoramento nos agentes para verificar sua disponibilidade. Quando agentes estiverem se comunicando através da rede é importante considerar que eles podem estar funcionando corretamente, mas não estão sendo capazes de se comunicar. Falhas na rede podem conduzir a *hosts* inalcançáveis tornando complexo distinguir entre a falha de uma agente e uma falha na rede.

O monitoramento é realizado através de troca de mensagens entre o Agente Sentinela e os agentes a serem monitorados. É importante ter a garantia da confiabilidade das mensagens enviadas. Ou seja, que a mensagem de *ping* foi realmente entregue ao destinatário, afastando a possibilidade de alarmes falsos causados pela perda de mensagens. No NIDIA, essa garantia de confiabilidade é realizada por [Oliveira and Abdelouahab, 2006]. Outro ponto importante é se ter uma rápida reação de recuperação quando esse tipo de falha for detectado.

Existem dois modelos de detecção de disponibilidade: *pull* e *push*.

No modelo *push*, um componente envia uma mensagem "*Eu estou vivo*" para um componente detector de falhas. Se a mensagem não é recebida em um certo intervalo de tempo, o componente detector de falhas deduz a falha no componente.

No modelo *pull*, um componente detector de falhas envia a mensagem "*Você está vivo?*" para os componentes a serem monitorados e fica aguardando, em um intervalo de tempo definido estaticamente ou dinamicamente, que o componente responda com a mensagem "*Eu estou vivo*". Caso não seja recebida uma resposta do componente, ele é

tido como falho.

O modelo *pull* simplifica o controle com que as mensagens são trocadas, embora o número de mensagens seja o dobro do modelo *push*. O modelo *push* tem como desvantagem acrescentar complexidade no componente a ser monitorado.

O modelo adotado pelo nosso mecanismo é o modelo *pull*, o Agente Sentinela envia periodicamente mensagens de"*Você está vivo?*" aos agentes do NIDIA e fica aguardando por mensagens de retorno. Se após um intervalo de tempo uma mensagem de retorno não é recebida, o Agente Sentinela envia um alerta ao Agente de Avaliação de Falhas informando que um agente no sistema não respondeu, e portanto foi considerado falho.

O Agente de Avaliação de Falhas analisa qual a melhor recuperação para este tipo de falha. Neste caso, um novo agente deve ser iniciado no mesmo *host* onde o agente que falhou estava executando. Uma solicitação de inicialização de um novo agente é repassada ao SRA.

A Figura 4.6 mostra o diagrama de seqüência da detecção de *crash* de um agente, usando a notação UML [UML, 2005]. Os passos dessa detecção são descritos a seguir.

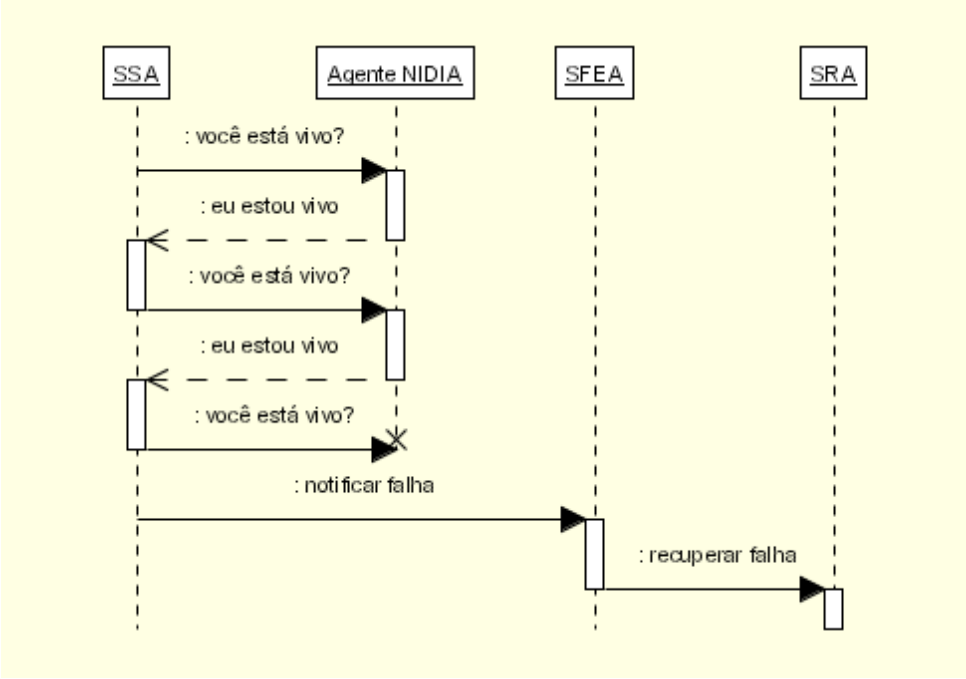

Figura 4.6: Diagrama de Sequência - Detecção de *Crash* de Agente

i. O Agente Sentinela periodicamente envia mensagens " *Você está vivo?*" aos agentes

do NIDIA;

ii. O agente NIDIA recebe essa solicitação e responde com mensagem "*Eu estou vivo*";

iii. Caso a resposta do agente NIDIA não seja recebida num intervalo de tempo especificado, o Agente Sentinela deduz que este agente falhou e envia uma mensagem de notificação ao Agente de avaliação de Falhas;

iv. O Agente de Avaliação de Falhas, por sua vez seleciona a melhor forma de recuperação para este caso de falha. Ele envia uma solicitação ao SRA para inicializar de um novo agente no *host* onde o que falhou estava executando.

A escolha pelo método *pull*, uma vez que este modelo utiliza o dobro de mensagens, se deve ao fato de ser mais simples de controlar e também ao fato de termos o componente detector de falhas, nesse caso, o Agente Sentinela localizado no mesmo *host* onde os agentes a serem monitorados estão executando. Portanto, não há sobrecarrega na rede com as mensagens de detecção. Isso evita o problema de falsas detecções devido a atraso das mensagens causadas pela sobrecarga ou outro problema na rede. A comunicação local é rápida e simples.

Para garantir a tolerância a falhas do próprio mecanismo, a detecção de *crash* dos agentes do mecanismo também deve ser provida. Os Agentes de Replicação e o Agente de Avaliação de Falhas também são monitorado pelo Agente Sentinela localizado no *host* onde estes estão executando.

E para os Agentes Sentinelas é utilizado a sua hierarquia para se realizar essa detecção. Ou seja, os Agentes Sentinelas são monitorados pelos seus agentes superiores na hierarquia. Por exemplo, os Agentes Sentinelas no nível *Host* são monitorados pelos Agentes Sentinelas no nível Domínio. E somente nesse monitoramento haverá comunicação através da rede.

É importante ressaltar que os agentes do mecanismo de tolerância a falhas, Agente de Replicação e Agente de Avaliação de Falhas, também serão monitorados pelo Agente Sentinela.

4.5.3 Detecção de Mudança na Estratégia de Replicação do Agente

Como a importância de um agente pode evoluir durante o curso da computação, é necessário dinamicamente e automaticamente adaptar o número de réplicas dos agentes, a fim de maximizar a confiabilidade e disponibilidade baseada nos recursos disponíveis.

O mecanismo visa tomar decisões para adaptar a estratégia de replicação dos agentes em reação ao comportamento do sistema. Obviamente, determinar o comportamento do sistema requer algum mecanismo de monitoramento. Esse monitoramento, como dito anteriormente, é desempenhado pelo Agente Sentinela.

O comportamento do sistema pode ser definido como o estado do sistema em um dado momento. Em tempo de execução, o Agente Sentinela observa um conjunto de informações nos agentes e nos *hosts*. E uma avaliação sobre essas informações é realizada. A partir dessa avaliação é possível determinar qual a melhor estratégia de replicação para cada agente do sistema.

Para determinar a importância de um agente necessitamos saber primeiramente qual a dependência dos demais agentes em relação a este. Vamos fazer uma análise das duas primeiras camadas do NIDIA: camada de monitoramento e camada de análise. Estas são consideradas as mais importantes camadas do sistema.

Na camada de monitoramento encontram-se os Agentes Sensores de Rede e os Agentes Sensores de *Host*. O funcionamento do sistema depende das informações geradas por estes dois agentes.

Na camada de análise encontram-se os agentes responsáveis pela detecção de um ataque. E, portanto, o funcionamento correto da detecção depende do funcionamento correto destes agentes.

Outra forma de definirmos a importância de um agente é através da quantidade de mensagens recebidas num intervalo de tempo. Quando um agente está sendo muito solicitado, isso significa que naquele momento ele é de grande importância para o sistema. E caso venha a falhar, isso acarretará graves conseqüências para a entrega do serviço do sistema.

4.6 Vantagens e Limitações

O mecanismo proposto aborda um aspecto que ainda se encontra bastante imaturo na área de sistemas de detecção de intrusão que é a tolerância a falhas. Dentro da literatura consultada, observou-se que os sistemas de detecção de intrusão multiagentes mencionam o fato de serem distribuídos como garantia de tolerância a falhas.

A natureza modular de um sistema multiagente dá a ele um certo nível inerente de tolerância a falhas, porém a natureza não deterministica dos agentes, o ambiente dinâmico e falta de um ponto de controle central torna impossível prever possíveis estados falhos e torna o comportamento do gerenciamento de falhas imprevisível [Fedoruk and Deters, 2002].

Considerar que com a falha de um agente, o sistema pode ainda continuar funcionando porque não há parado total do sistema é uma afirmação verdadeira. No entanto, um sistema multiagente também significa que o sistema passa a ter os vários pontos de falhas, que são os agentes.

Conseqüentemente, é necessário tratar explicitamente esse aspecto em um sistema multiagente, e principalmente em um sistema de detecção de intrusão, que é um sistema cuja disponibilidade e confiabilidade são características muito importantes.

O mecanismo de tolerância a falhas proposto utiliza replicação de agentes para prover a tolerância a falhas e também o balanceamento de carga entre os agentes. Ele utiliza monitoramento dos agentes para detectar alguns tipos de falhas, tais como *crash* e agentes maliciosos, e ainda provê uma recuperação adequada para cada tipo de falha.

O mecanismo de tolerância a falha proposto visa abranger diferentes tipos de falhas no sistema de detecção de intrusão multiagente (NIDIA), no entanto ele somente trata de falhas nos agentes do sistema. Ele não detecta quando há falhas nos *hosts* ou na comunicação. A falha de um *host* tem como consequencia a falha de todos os agentes executando nele. A falha na comunicação impede que os agentes possam trocar mensagens e sendo assim o funcionamento do sistema como um todo fica comprometido.

A Tabela 4.1 apresenta uma análise comparativa entre os sistemas de detecção de intrusão baseados em agentes apresentados no capítulo 2 e o NIDIA.

Essas falhas ainda não exploradas são objetos de futuras pesquisas dentro do NIDIA.

Tabela 4.1: Comparação entre SDIs Multiagentes e o NIDIA

Projeto	Fonte de Informação	Método de Detecção	Reação	Mobilidade	T. F.
Hegazy et al.	Rede	Assinatura	Ativo	Não	Não
AAFID	Rede e *Host*	Assinatura	Passivo	Não	Não
MADIDS	*Host*	-	-	Sim	Não
Ramachadram	*Host*	-	Ativo	Sim	Não
Zhicai	Rede e Host	Assinatura e Anomalia	-	Sim	Não
SPARTA	Rede e Host	Assinatura	Passivo	Sim	Não
NIDIA	Rede e Host	Assinatura e Anomalia	Ativo	Sim	Sim

4.7 Conclusões

Apresentou-se neste capítulo um mecanismo de tolerância a falhas para o NIDIA baseado em uma sociedade de agentes inteligentes, realizando o monitoramento e recuperação do sistema quando ocorrer falhas acidentais ou maliciosas.

A função de cada agente foi mostrada em detalhes. A replicação dos agentes dentro do sistema foi apresentada. E alguns cenários de detecção foram apresentados: detecção de agentes falhos, detecção de agentes maliciosos e ainda mudanças na estratégia de replicação de grupos de agentes.

5 Implementações Parciais e Resultados

O NIDIA foi inicialmente implementado na plataforma de agentes ZEUS [Nwana et al., 1999]. O ZEUS é uma plataforma de desenvolvimento de agentes que suporta o gerenciamento e a comunicação de agentes autônomos e estáticos em Java.

No entanto, com novas pesquisas e adição de novas funcionalidades ao NIDIA, houve necessidade de utilização de uma plataforma que oferecesse algumas características que não estão inseridas no escopo da plataforma ZEUS, como por exemplo, migração de agentes. Conseqüentemente, uma nova plataforma de desenvolvimento tornou-se necessária. Os membros do projeto NIDIA optaram pela plataforma JADE.

Este capítulo aborda a implementação do NIDIA e do mecanismo de tolerância a falhas proposto ao mesmo, utilizando para o desenvolvimento dos agentes a plataforma JADE.

Primeiramente, uma breve descrição da plataforma JADE é apresentada. Um resumo das implementações do NIDIA nessa plataforma é descrito.

Finalmente, a implementação do mecanismo de tolerância a falhas é apresentando. Detalhes de cada um dos seus componentes, agentes e base de dados, são mostrados.

5.1 Plataforma JADE

***J**ava **A**gent **DE**velopment Framework* (JADE) é um *framework* para desenvolver aplicações baseadas em agentes [JADE, 2005, Bellifemine et al., 2005], conforme as especificações da *Foundation for Intelligent Physical Agents* (FIPA) [FIPA, 2005] para sistemas multiagentes inteligentes interoperáveis. Ele é um *software* livre e completamente implementado em Java, portanto, independente de plataforma [de Oliveira et al., 2001]. JADE suporta mobilidade intraplataforma e clonagem de agentes.

JADE lida com aspectos que não são peculiares da natureza interna do agente

e que são independentes de aplicações, tais como, transporte de mensagem, codificação e ciclo de vida do agente. Ele também oferece um conjunto de ferramentas que dão suporte as fases de implementação, depuração e implantação do sistema.

Uma característica importante dessa plataforma é que ela pode ser distribuída por vários *hosts*, cada um deles executando apenas uma JVM (*Java Virtual Machine*). Os agentes são implementados como *threads* Java e inseridos dentro de repositórios de agentes chamados de *containers*. Estes *containers* provêm todo o suporte para a execução do agente e representam o ambiente de execução das aplicações de agentes [da Silva, 2005]. A Figura 5.1 apresenta a estrutura da plataforma de agentes JADE distribuída por vários *hosts*.

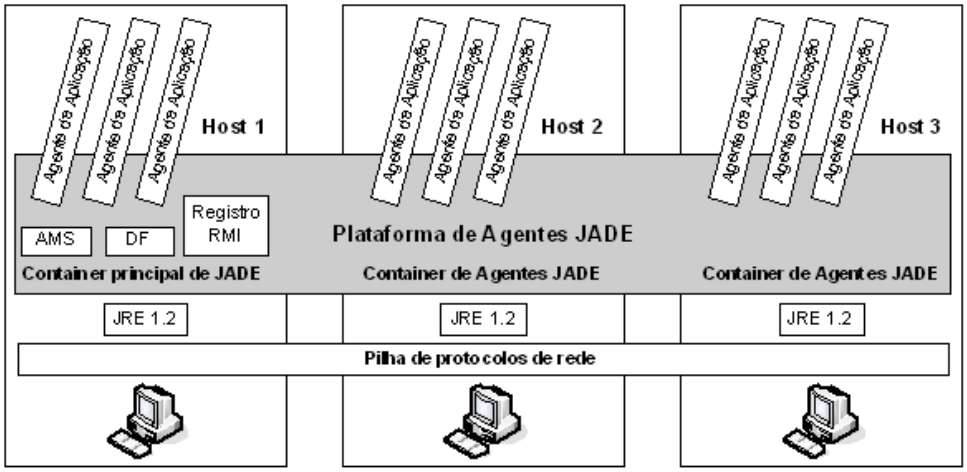

Figura 5.1: Plataforma de Agentes JADE distribuída por vários *containers*

Uma plataforma pode conter vários *containers*, que podem estar localizados em *hosts* distintos. Em cada plataforma existe um *container* especial, denominado *Main-Container*, onde residem os seguintes componentes:

- Sistema gerenciador de agentes (*Agent Management System* - AMS): é responsável pela supervisão de acesso e uso da plataforma. O AMS mantém informações do agente, tais como, identificador (AID - *Agent Indentifier*) e estado dos agentes. Todos os agentes se registram no AMS. Em uma plataforma, existe apenas um AMS que é criado automaticamente;

- Facilitador de diretórios (*Directory Facilitator* - DF): é o serviço de páginas amarelas. Também é criado automaticamente;

- Registro RMI (*Remote Method Invocation Registry*): é um servidor de nomes que

Java utiliza para registrar e recuperar referências a objetos através do nome. Ou seja, é o meio que JADE utiliza para manter referências aos outros *containers* de agentes que se conectam a plataforma.

Agentes em JADE são processos que representam entidades autônomas independentes. Eles possuem uma identidade e a capacidade de se comunicar com outros agentes para atingir os seus objetivos de execução [JADE, 2005]. Um agente pode realizar múltiplas tarefas e conversações de forma simultânea. Ele controla completamente sua linha de execução e decidi quando deve ler as mensagens recebidas e quais mensagens serão lidas.

A Figura 5.2 ilustra a estrutura de um agente simples em JADE. O agente é composto da classe principal *Agent* e de uma classe *Behaviour*. A classe *Agent* facilita o desenvolvimento de agentes porque provê todas as características necessárias para realizar as interações básicas com a plataforma, tais como métodos para registro, configuração e gerenciamento remoto do agente. Ela possui o método *setup*(). É nele que deve ser implementado o código de configuração inicial do agente, tais como construir e adicionar comportamentos, registrar ontologias, registrar o agente no DF, etc.

Behaviour é uma classe usada para modelar uma tarefa genérica do agente, ou seja, seu comportamento. Cada serviço ou funcionalidade de um agente deve ser implementado como um ou mais comportamentos. O método *action*() da classe *Behaviour* descreve as ações que são executadas pelos comportamentos. Definir comportamentos de um agente é praticamente definir que ações ele deve tomar em determinadas situações. Ou seja, é uma das funções mais importantes no desenvolvimento de sistemas multiagentes, pois é nelas que vão ser definidos os comportamentos que esses agentes terão no ambiente.

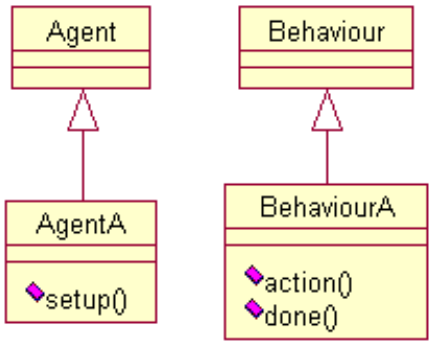

Figura 5.2: Classes do agente JADE

Todo agente possui uma identificação única, denominada de AID (*Agent Identidier*). Esse identificador é constituído por: um nome globalmente único que deve obedecer à estrutura *localname*@*hostname*:*port*/JADE; um conjunto de endereços de agentes; e um conjunto de solucionadores (uma vez que cada agente herda os endereços de transporte de sua plataforma de origem), ou seja, páginas brancas nas quais o agente é registrado [da Silva, 2005].

Outra característica importante dos agentes JADE é a capacidade de comunicação. O modelo de comunicação utilizado é a passagem de mensagem assíncrona. Cada agente possui uma fila privativa de mensagens onde são depositadas mensagens postadas por outros agentes. As mensagens trocadas têm um formato especificado pela Linguagem de Comunicação de Agentes (ACL - *Agent Communicaton Language*) definida pelo padrão internacional da FIPA para interoperabilidade de agentes.

JADE possui um conjunto de agentes de serviços de sistema disponível através da plataforma. Esses agentes são ferramentas de suporte ao desenvolvedor e lidam com aspectos que não fazem parte do agente em si e que são independentes da aplicação, tais como: transporte de mensagens, codificação e interpretação de mensagens, ciclo de vida dos agentes, páginas amarelas e páginas brancas. Esses agentes de serviço são:

- *Remote Management Agent* (RMA) : funciona como uma console gráfica para o controle e gerenciamento da plataforma. Ela é usada para monitorar e administrar o estado de todos os componentes da plataforma distribuída, incluindo agentes e *containers*. Pode existir mais de um RMA na mesma plataforma, mas apenas um pode existir no mesmo *container*;

- *DummyAgent*: é uma ferramenta gráfica de monitoramento e *debugging* para agentes. Com ela é possível criar e enviar mensagens ACL para outros agentes e também listar todas as mensagens ACL enviadas e recebidas por um agente, com informações detalhadas. Além disso, é possível salvar essa lista de mensagens em disco e recuperá-la depois;

- *Sniffer Agent*: é uma ferramenta utilizada para interceptar as mensagens ACL e exibir a conversação entre agentes através de uma notação similar a diagramas de seqüências da UML. Quando um usuário decide fazer um *sniff* em um agente ou grupo de agentes, toda mensagem direcionada a este agente/grupo de agentes ou

vinda destes, é rastreada e disponibilizada. O usuário poderá ver todas mensagens e salvá-las em disco para uso posterior.

- *Introspector Agent*: é uma ferramenta que permite monitorar o ciclo de vida de um agente, suas trocas de mensagens e seus comportamentos em execução. Além disso, essa ferramenta permite o controle da execução do agente, no caso, uma execução passo-a-passo.

- *DF GUI*: é uma ferramenta que provê o serviço de páginas amarelas da plataforma. Ela permite que operações de registro, cancelamento de registro, alteração de registro e busca por agentes e serviços sejam realizadas. Em cada plataforma existe pelo menos um DF.

5.2 Implementação do Protótipo

A migração de plataforma foi a primeira etapa do trabalho de implementação. O protótipo atual do NIDIA, na plataforma JADE, utilizou a implementação feita por [Lima, 2002] para o Agente Sensor de Rede; as implementações de [Dias, 2003] para os agentes de Monitoramento do Sistema (SMA) e de Avaliação de Segurança do Sistema (SEA).

O Agente Controlador Principal (MCA) e o Agente BAM (BA) implementados parcialmente por [Pestana, 2005] e a integração realizada entre o protótipo de [Pestana, 2005] e o protótipo desenvolvido por [Oliveira, 2005], foram também migrados para a nova plataforma.

Os agentes citados acima podem ser observados na Figura 3.1.

As tarefas realizadas pelos agentes do NIDIA foram transformadas em comportamentos. Criou-se uma ontologia para prover a troca de mensagens entre os agentes. A ontologia define um vocabulário para a troca de mensagens entre os agentes.

Dessa forma, criamos um ambiente de execução do NIDIA funcionando, mesmo que parcialmente, para que as implementações e validações do mecanismo junto ao NIDIA pudessem ser efetivamente realizadas.

As implementações de alguns agentes do NIDIA e do mecanismo de tolerância a falhas proposto são apresentados a seguir.

5.2.1 Implementação do Agente de Segurança Local - LSIA

O primeiro agente a ser implementado foi o LSIA, que pode ser observado na Figura 3.1. Ele é o primeiro agente do NIDIA a ser iniciado no sistema. LSIA é o agente responsável pelo gerenciamento da sociedade de agentes e pela interface entre o NIDIA e o administrador de segurança.

O LSIA utiliza um arquivo de configuração denominado "*agents.txt*" para ativar os demais agentes na plataforma de agentes JADE. Dessa forma, é possível ativar de forma centralizada todos os agentes, sendo que estes estarão distribuídos pela rede. O administrador de segurança do sistema escolhe quais as funcionalidades do NIDIA que são necessárias de acordo com a política de segurança da organização. Por exemplo, é possível omitir alguns agentes da camada de reação ou escolher se deseja ter agentes sensores de rede ou de *host* ou ambos. A desvantagem dessa abordagem é que ela obriga o admninistrador de segurança a conhecer todos os agentes do NIDIA e conhecer suas funcionalidades.

A Figura 5.3 mostra um exemplo do arquivo "*agents.txt*".

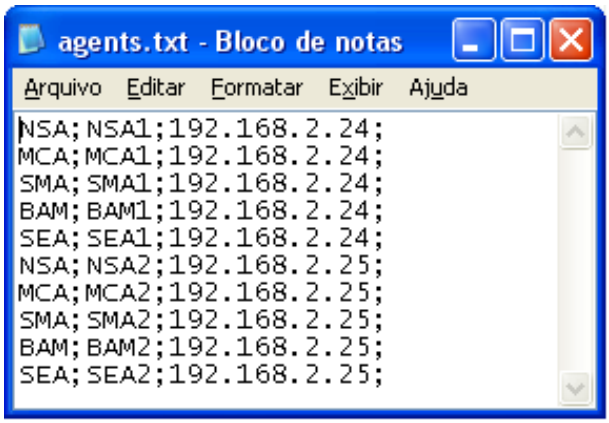

Figura 5.3: Arquivo agents.txt

O arquivo "*agents.txt*" contém três tipo de informações (elas estão separadas por ";") que são descritas a seguir:

1. Tipo do agente: representa os agentes mencionados no capítulo 3 e significa para o sistema o serviço que é oferecido pelo agente. Exemplo: SMA, SEA,BAM, etc;

2. Nome do agente: é o nome pelo qual os agentes serão conhecidos dentro da plataforma de agentes;

3. *Host* do agente: endereço do *host* onde o agente deve ser iniciado. Pode ser o nome ou endereço IP do *host*. No exemplo mostrado na Figura 5.3 foi utilizado o endereço IP do *host*;

Para cada agente no arquivo, o seguinte procedimento é realizado:

1. O tipo do agente é guardado e é verificado se existe um Agente de Replicação para aquele tipo de agente. Um Agente de Replicação é criado para cada tipo de agente que for ativado no sistema;

2. O endereço do *host* é guardado e é verificado se existe um Agente Sentinela naquele *host*. Caso não exista, um Agente Sentinela é iniciado naquele *host*. Um Agente Sentinela será iniciado para cada *host* diferente que "hospedar" um agente NIDIA;

3. O agente é iniciado no *container* existente no *host* indicado;

4. Uma mensagem para registrar esse novo agente é enviado ao Agente Sentinela desse *host*. O Agente Sentinela possui uma lista de todos os agentes que ele deve monitorar.

Após esse procedimento, o LSIA inicia o Agente de Avaliação de Falhas. Este agente residirá no *Main-Container* da plataforma JADE, assim como todos os Agentes de Replicação.

A cada novo agente iniciado, ele é registrado no *Directory Facilitator* (DF), onde são cadastrados os serviços oferecidos pelos agente. O nome do serviço dos agentes é representado pelo tipo de cada agente. Os agentes são registrados no DF utilizando as classes providas pelo JADE: *ServiceDescription*, *DFAgentDescription* e *DFService*. O Código 5.2.1 mostra como é feito esse cadastro.

Logo após cada agente recebe seus comportamentos. Estes procedimentos são comuns a todos os agentes e é realizado pela classe *Agent.java*, comum a todos os agentes, com exceção do LSIA.

Na classe Agent.java está inserido o método que atribui os comportamentos aos agentes. Cada agente receberá os comportamentos que são adequados para o seu tipo, como pode ser observado no Código 5.2.2.

O tipo do agente (*typeAgent*) determina quais os comportamentos que ele deve possuir. Na linha 5 é checado se o agente é o Agente de Replicação ou uma réplica. Na

Código 5.2.1 *método registerService()*

```
1  public void registerService (Agent agente) {
2
3                  DFAgentDescription dfd = new DFAgentDescription ();
4                  dfd.setName (agente.getAID());// nome do agente
5                  ServiceDescription sd = new ServiceDescription ();
6                  sd.setType (agente.getLocalName()); // tipo do serviço
7                  sd.setName (agente.getLocalName()); // nome do serviço
8                  dfd.addServices (sd);
9
10                 try {
11                         DFService.register (agente, dfd);
12                 } catch (FIPAException fe) {
13                         System.err.println (agente.getLocalName() +
14                          " não registrado no DF." +
15
16                 }
17 }
```

Código 5.2.2 *método setup() na classe Agents.java*

```
1
2  ...
3
4  else if (typeAgent.compareTo("SMA") == 0){
5          if (despatcher.compareTo("D") == 0){
6                  // Agente de Replicação
7                  addBehaviour(new DABehaviour (this));
8          }
9          else {
10
11                 addBehaviour(new SMABehaviour (this));
12         }
13         addBehaviour(new AgentReceivePing (this));
14
15 }
16
17 ...
18
19
20
21
```

linha 13, o agente recebe o comportamento *AgentReceivePing*. Este comportamento é comum a todos os agentes. Ele aguarda uma mensagem de *ping* do Agente Sentinela e responde a mensagem.

As classes pertencentes ao agente LSIA são mostradas a seguir:

- LSIAGui.java: Interface gráfica do LSIA;
- LSIAAgent.java: Esta classe tem as seguintes funções: criar e iniciar repositório de dados; atribuir os comportamentos do LSIA;
- LSIABehaviourAddAgent.java: Esta classe inicia os agentes do NIDIA, a partir do arquivo de configuração e os agentes do mecanismo de tolerância a falhas;
- LSIABehaviourStartAgent: Esta classe inicia um agente em um determinado *host* através da solicitação de outro agente. Ele encaminha uma solicitação de criação ao

AMS do JADE.

5.2.2 Implementação do Agente Sentinela - SSA

O Agente Sentinela é um agente essencial na arquitetura. As informações coletadas e geradas pelo Agente Sentinela são utilizadas pelos demais agentes na arquitetura do mecanismo de tolerância a falhas. Sua principal função consiste em monitorar informações previamente estabelecidas.

Para cada *host* que pertencer a plataforma de agentes haverá um Agente Sentinela. Quando um Agente Sentinela é iniciado, essa informação é guardada.

Ele coleta informações de *host* através do comportamento implementado na classe *SSABehaviourCapabilities.java*. Esse comportamento é da classe *TickerBehaviour*. Essa classe implementa um comportamento executando periodicamente tarefas específicas.

A classe captura informações da *host*: memória utilizada, memória total, memória disponível, espaço em disco total, espaço em disco utilizado, espaço em disco disponível e uso do processador.

Ele possui também a função de monitorar a disponibilidade dos agentes do NIDIA. Através do comportamento *SSABehaviourMsgPing.java*, que é do tipo *TickerBehaviour*, ele envia mensagens sem conteúdo para agentes executando no seu *host*. Aguarda um retorno e assim pode determinar se estes agentes estão vivos (executando) ou não.

Cada Agente Sentinela possui uma lista dos agente que ele deve monitorar. A cada intervalo de tempo ele consulta essa lista e envia mensagens para todos os agentes. Caso alguma agente não responda ele envia uma mensagem de alerta ao Agente de Avaliação de Falhas.

As classes pertecentes ao Agente Sentinela são descritas a seguir:

- SSABehaviourMsgPing: Envia mensagem aos agentes localizados no seu *host*;
- SSAWakerBehaviourPing: Aguarda a mensagem de retorno dos agentes para saber se estão ativos. Quando detecta que existe agente inativo envia alerta para Agente de Avaliação de Falhas;

- SSABehaviourRegister: Comportamento que cadastra os agentes que o SSA deve monitorar;
- SSABehaviourCapabilities: Comportamento que recupera as informações do *host*.

5.2.3 Implementação do Agente de Avaliação de Falhas - SFEA

O Agente de Avaliação de Falhas é responsável por associar cada tipo de falha com sua adequada recuperação.

Este agente possui um comportamento que recebe informações do Agente Sentinela. Na informação recebida são passadas alguns dados do tipo: a falha detectada, o agente que falhou ou provocou a falha, entre outras informações dependendo de cada tipo de falha.

Baseado nessas informações, o Agente de Avaliação de Falhas determinará qual a melhor estratégia de recuperação dependendo das informações recebidas, como pode ser observado no Código 5.2.3.

Código 5.2.3 *Recuperação adequada para cada tipo de falha*

```
...
if (ce instanceof MessageSFEA) {

        MessageSFEA mensagem = (MessageSFEA) ce;
        typeFailure = mensagem.gettypeFailure();


        if (!(typeFailure.indexOf("AGENTCRASH") < 0)) {
                ...
                // deve iniciar um novo agente do mesmo tipo e com mesmo nome

                startNewAgent(typeAgent, nameAgent, IPAgent, despatcher );


        } else if (!(typeFailure.indexOf("AGENTMAL") < 0)) {
                ...
                // excluir agente malicioso e iniciar novo agente

                CreateNewAgent( nameAgent);


        } else if (!(typeFailure.indexOf("CHANGEREPLICATION") < 0)) {
                ...
                // alterar a estratégia de replicação do agente

                ChangeReplication(typeAgent)


        }

}

...
```

5.2.4 Implementação do Agente de Replicação

O Agente de Replicação é criado automaticamente pelo sistema. Para cada tipo diferente de agente é criado um Agente de Replicação. Todos os Agentes de Replicação

são iniciados no "*Main-Container*".

Os agentes do NIDIA não têm conhecimento da existência do Agente de Replicação. No entanto, quando uma mensagem é enviada de um agente para outro no sistema, ela é sempre recebida pelo Agente de Replicação que faz a distribuição da mensagem para o seu grupo de agentes. Ele consulta os *hosts* que estão as réplicas do seu grupo e a partir de informações coletadas no *host* ele decide qual a réplica que está mais disponível e portanto deve receber a mensagem.

5.2.5 Base de Dados de Perfis

A Base de Dados de Perfis (PRDB) foi implementada como um repositório de metadados [Lopes, 2006]. A Figura 5.4 mostra o diagrama de classe da Base de Dados de Perfis.

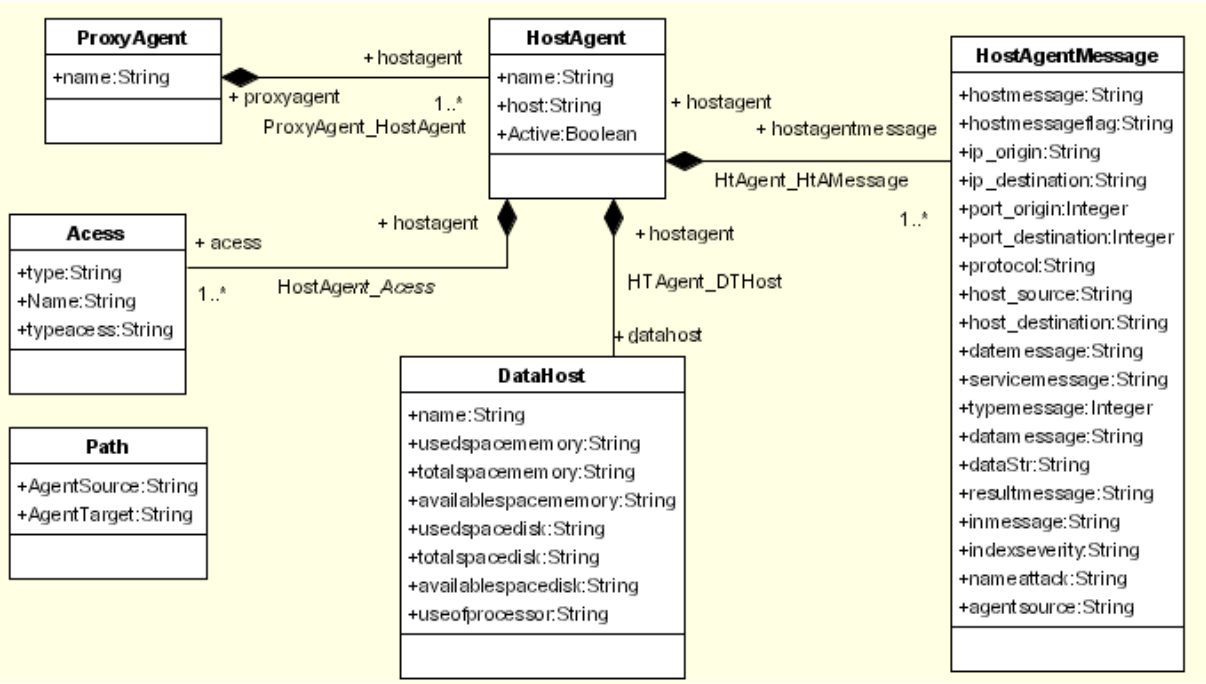

Figura 5.4: Diagrama de Classes da PRDB

A classe *ProxyAgent* representa o Agente de Replicação. Esse agente está associado às réplicas que ele gerencia representadas pela classe *HostAgent*. A cada réplica de um agente estão associadas as mensagens que foram encaminhadas a ela pelo Agente de Replicação. As mensagens são representadas pela classe *HostAgentMessage*. A cada réplica também está associada a classe *DataHost*, nessa classe estão todas as informações do *host* onde estão executando àquelas réplicas.

A classe *Path* é utilizada para a detecção de agentes maliciosos. Ele representa

o fluxo de mensagens dentro do NIDIA. Ou seja, nela é armazenado que agente tem permissão de enviar mensagem para um determinado agente.

Essa base de dados tem seus elementos criados toda vez que o NIDIA for iniciado. Isso porque as informações armazenadas são dependentes da execução do sistema. A criação do repositório é realizada pelo agente LSIA.

5.3 Resultados

Os testes foram realizados em três *hosts*, cujos IPs são: 192.168.2.24, 192.168.2.25, 192.168.2.26. A plataforma JADE foi distribuída entre esses *hosts*. A *host* 192.168.2.26 hospeda o "*Main-Container*" e os demais hosts hospedam os *containers* comuns. A seguir mostraremos os resultados obtidos com os testes realizados em laboratórios

5.3.1 NIDIA na Plataforma JADE

O agente LSIA é iniciado no sistema e fica executando no "*Main-Container*". Ele inicia os demais agentes através do arquivo "*agents.txt*" mostrado anteriormente.

Podemos observar através da agente de serviço RMA, disponibilizado pelo JADE, como os agentes foram distribuídos. A Figura 5.5 mostra através do RMA e os agentes do NIDIA na plataforma.

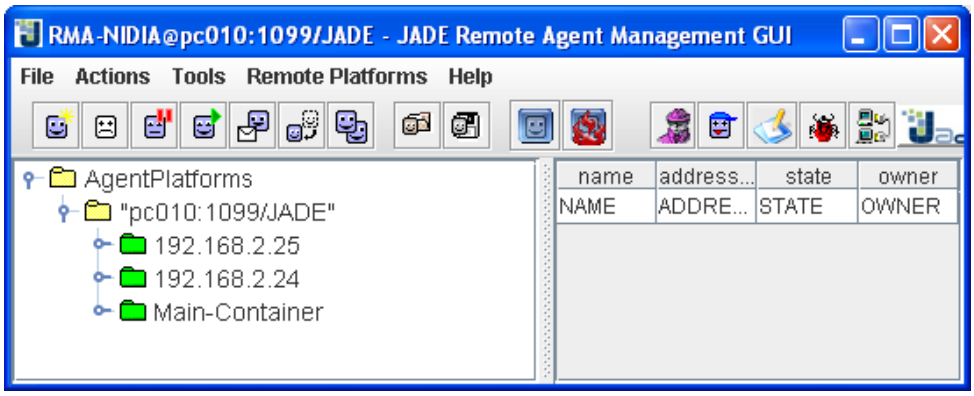

Figura 5.5: NIDIA na plataforma JADE

As Figuras 5.6 e 5.7 mostram, respectivamente, os agentes do NIDIA nos *containers* comuns e no *Main-Container*.

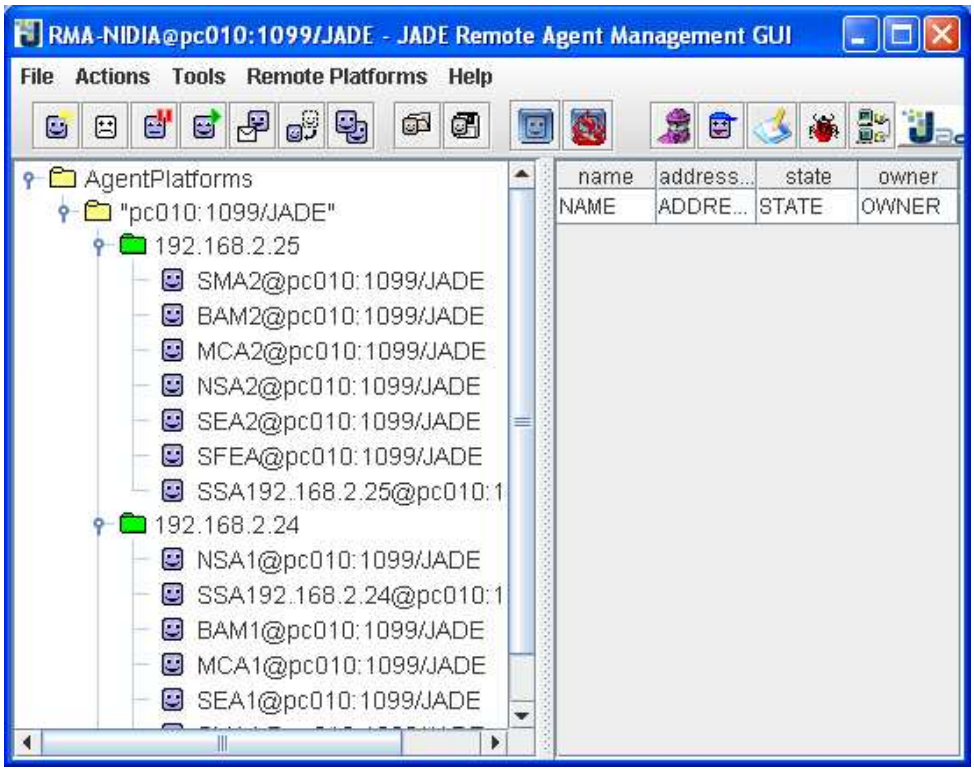

Figura 5.6: Agentes NIDIA em containers comuns

5.3.2 Detecção de Falhas - Agentes Maliciosos

Na Base de Dados de Perfis, há um classe denominada *Path* que determina a integração entre os agentes. Ou seja, nela está armazenado o fluxo de mensagens que deve ser seguido pelo sistema. Ela possui as seguintes informações: Agente Origem e Agente Destino.

Uma classe denominada Comunicacao.java foi criada para fazer a checagem de fluxo de mensagem correta entre os agentes. Um agente ao receber uma mensagem de um outro agente utiliza um método chamado *checkPath()* para verificar se o agente emissor está autorizado a lhe enviar mensagem. Veja o Código 5.3.1, onde é dado um exemplo dessa checagem.

Caso não seja autorizada aquela comunicação, o agente que detectou essa falha informa ao Agente Sentinela existente no seu *host* sobre esse acontecimento. Isto evita que agentes não autorizados enviem mensagens pelo sistema causando sobrecarga e funcionamento incorreto no agente receptor.

Como exemplo, temos: o Agente Sensor de Rede somente envia mensagens ao Agente de Análise do Sistema. Caso algum outro agente venha a receber uma mensagem do Agente Sensor de Rede, deve imediatamente alertar ao Agente Sentinela.

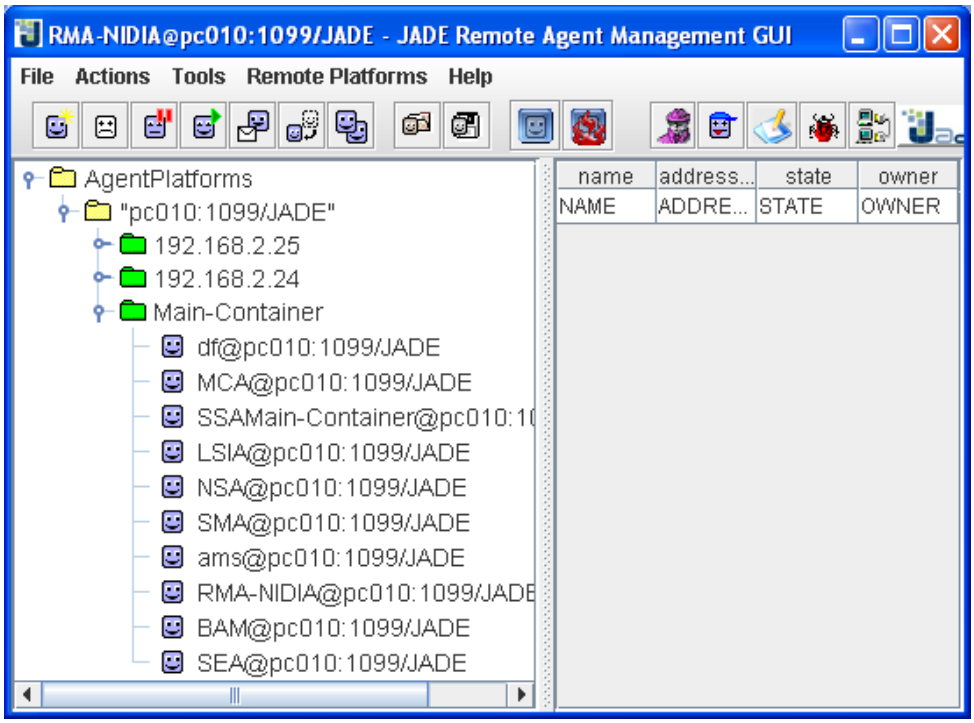

Figura 5.7: Agentes NIDIA no Main-Container

Código 5.3.1 *Checagem do fluxo de mensagem no NIDIA*

```
1  private boolean receiveMessage()  {
2
3      ...
4      mt =  MessageTemplate.MatchPerformative( ACLMessage.REQUEST );
5      ACLMessage msg = myAgent.receive (mt);
6      if (msg != null) {
7         try {
8            ContentElement ce = manager.extractContent (msg);
9             nameSource =                                    msg.getSender().getName().subs
10            nameTarget = myAgent.getName().substring(0,myAgent.getName().indexOf("@"));
11
12            if (!com.checkPath(nameSource,nameTarget)){
13               // informar SSA
14               System.out.println("Origem inválida");
15               sendMessageSSA(nameSource);
16            }else{
17
18                 if (ce instanceof Message) {
19                         Message mensagem = (Message) ce;
20                 ...
21
22
23
24
```

5.3.3 Detecção de Falhas - *Crash* de Agentes

Testes para a detecção de *crash* de agentes foram realizados acompanhando mensagens de execução do sistema. Veja a seguir, na Figura 5.8, o resultado quando não há agentes inativos.

Para testarmos um agente inativo utilizamos o agente de serviço RMA do JADE para matar (*kill*) um agente do NIDIA. Veja o exemplo a seguir:

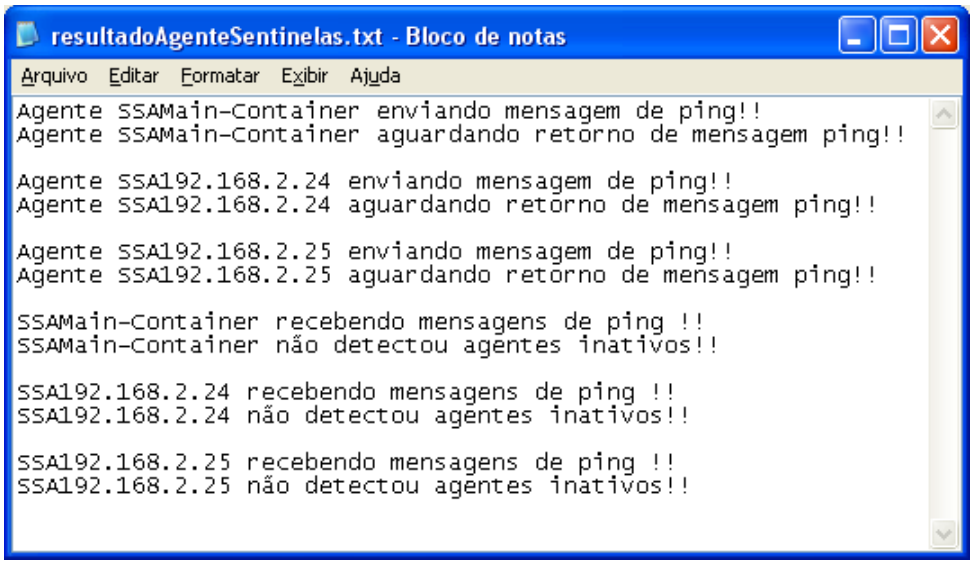

Figura 5.8: Arquivo com resultado de detecção de crash de agentes

A Figura 5.9 mostra o agente SEA2 sendo excluído da plataforma JADE.

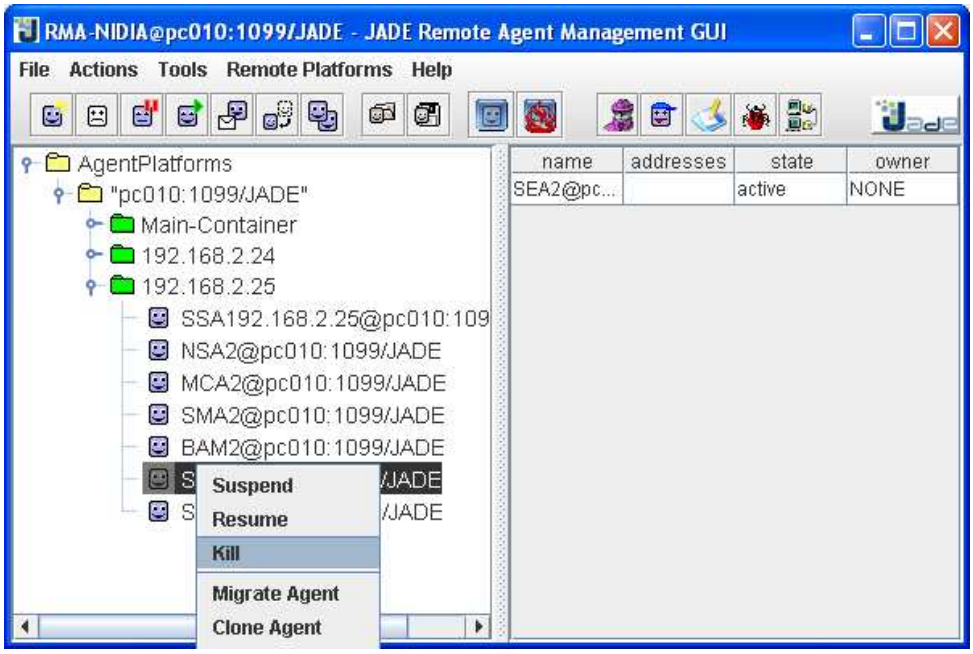

Figura 5.9: Falha do Agente SEA2 através do RMA

JADE chama o método *takedown()* automaticamente quando um agente morre. Esse método é utilizado para exluir o registro do agente que falhou no DF, evitando erros na criação de um novo agente com o mesmo nome do agente eliminado. Um vez que todos os agentes devem ser registrados no DF. A Código 5.3.2 mostra como é implementado o método *takedown()*.

A Figura 5.10 mostra as mensagens geradas após o agente SEA2 ter sido excluído.

Código 5.3.2 *Método takedown() na classe Agent.java*

```
protected void takeDown()   {

        System.out.println("Agente " + getLocalName() + " está fazendo shutting down.");
        try {
                DFService.deregister(this);
                System.out.println("Agente : " + getLocalName() + "falhou !!");
                System.out.println("Excluindo registro do agente : "
                                + getLocalName() + " no AMS!");
        }
        catch (Exception e) {
                System.err.println("Erro na exclusão do registro!");
        }
}
```

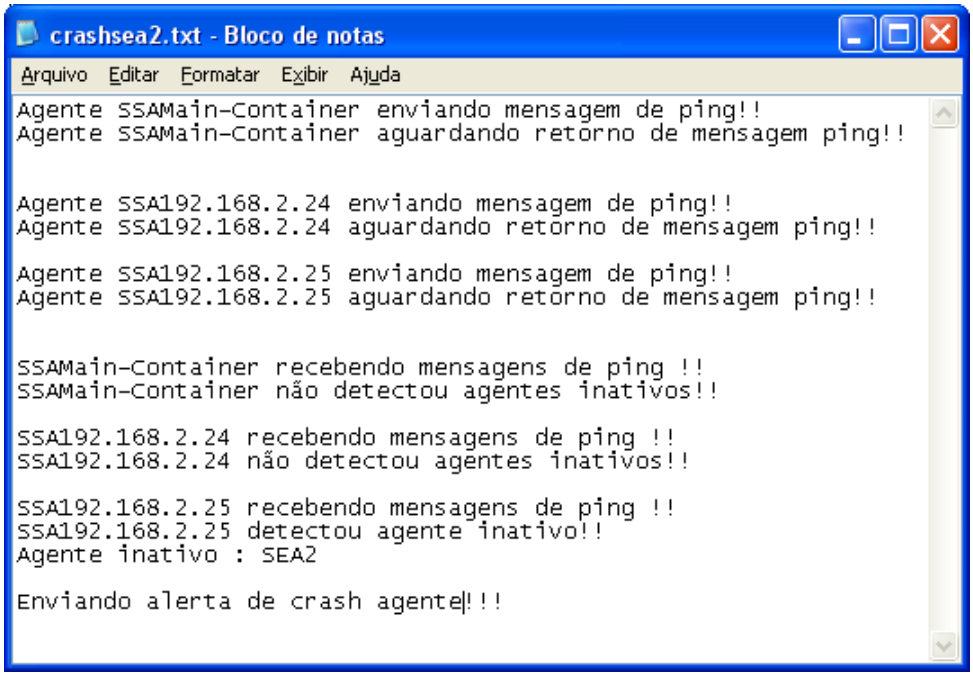

Figura 5.10: Arquivo com resultado de detecção de crash do agente SEA2

Em todos os testes o agente que foi eliminado teve sua criação sendo efetivada novamente.

6 Conclusões e Trabalhos Futuros

Este capítulo apresenta as contribuições deste trabalho, conclusões sobre os resultados alcançados e sugestões para trabalhos futuros.

6.1 Contribuições do Trabalho

Alguns sistemas de detecção de intrusão atuais, mesmo sendo adotado em um número crescente de instituições, não apresentam soluções para problemas como a sua própria segurança, a privacidade das informações trocadas entre seus módulos e a autenticidade desses módulos [Campello et al., 2001].

A falta de mecanismos de tolerância a falhas em sistemas de detecção de intrusão pode conduzir a sérios problemas e reduzir o escopo da sua aplicabilidade. Um sistema de detecção de intrusão deve ser tolerante a falhas para resistir a subversão[1], ou seja, ele deve ser capaz de monitorar a si mesmo e julgar se está corrompido. Caso considere que esteja corrompido, ele deve ser capaz de se recuperar. Além disso, deve também ser capaz de detectar falhas de seus componentes e providenciar uma recuperação adequada aos mesmos, continuando a oferecer seus serviços.

A principal contribuição deste trabalho foi o desenvolvimento de um mecanismo de tolerância a falhas para o NIDIA (um sistema de detecção de intrusão baseado em agentes). O mecanismo utiliza-se de duas abordagens para prover a tolerância a falhas: a replicação de agentes e o monitoramento do sistema (monitoramento de agentes e *hosts*). O monitoramento tem como objetivo detectar falhas, tanto acidentais quanto maliciosas. Uma recuperação adequada para cada falha detectada é também provida.

O mecanismo de tolerância a falhas foi desenvolvido como uma sociedade de agentes. A tarefa de replicação, de monitoramento e de recuperação do sistema foi distribuída entre os agentes da sociedade. O mecanismo tem as seguintes vantagens: i) arquitetura simples, de fácil compreensão e implementação; ii) não houve necessidade de grandes alterações nos comportamentos (funcionalidades) dos agentes do NIDIA.

Um protótipo do mecanismo de tolerância a falhas foi implementado. Para que esse protótipo pudesse ser avaliado, alguns testes em laboratório foram realizados.

[1] Ato ou efeito de transformar o funcionamento normal ou considerado bom de alguma coisa.

Algumas ferramentas disponibilizadas pela plataforma JADE foram utilizadas para a realização destes testes.

Este trabalho também propôs uma nova arquitetura para o NIDIA, baseada nas funcionalidades dos agentes existentes. A migração da plataforma de agentes ZEUS para plataforma JADE foi realizada. Os agentes do NIDIA, desenvolvidos por pesquisas anteriores, foram migrados para a nova plataforma.

6.2 Considerações Finais

Neste trabalho constatou-se a importância da aplicação de técnicas de tolerância a falhas em sistemas de detecção de intrusão. No entanto, pode-se constatar também que a maioria dos sistemas de detecção atuais não trata o problema da tolerância a falhas com a devida importância

Pouca literatura sobre tolerância a falhas em sistemas de detecção de intrusão foi encontrada, o que não nos permitiu realizar uma adequada comparação do nosso trabalho. Dessa forma, focamos nossas pesquisas e comparações em tolerância a falhas de sistemas multiagentes de um modo geral.

Constatou-se ainda que os sistemas multiagentes, em sua maioria, utilizam-se de redundância para prover tolerância a falhas.

A utilização da plataforma de agentes JADE tornou possível a realização de novas tarefas dentro do NIDIA, como por exemplo, a migração de agentes para a realização do balanceamento de carga do sistema. As ferramentas disponibilizadas por essa plataforma também foram de essencial importância nas fases de implementação e testes.

Outras pesquisas estão em andamento para o projeto NIDIA e tem como finalidade propor soluções para a comunicação segura e confiável entre os agentes do sistema, a utilização de mensagens baseadas em XML entre os agentes, a utilização das bases de dados em XML e a utilização do NIDIA de forma remota.

6.3 Trabalhos Futuros

Para a continuidade deste trabalho e sugestão para trabalhos futuros nessa área, podemos citar:

- A implementação de um algoritmo mais adequado de balanceamento de carga para a distribuição de mensagens entre os agentes de um grupo de réplicas;

- A otimização da inferência da importância do agente para o processamento do sistema para que a mudança de estratégia de replicação do grupo de agentes possa ser a mais adequada possível;

- Utilizar o modelo *push* para a detecção de falha de *crash* de agentes e comparar com o modelo *pull* que foi implementado;

- A implementação de uma interface gráfica que permita o administrador de segurança acompanhar todos os procedimentos inicialização, detecção de falhas, recuperação do sistema e outras funcionalidades NIDIA.

- Expandir o mecanismo para que este possa ser tolerante a falha de *hosts*, ou seja, quando ocorrer a falha de um *host* uma ação de recuperação dos agentes que estavam executando naquele possa ser providenciada. A falha de comunicação é outro problema que deve ser abordado por trabalhos futuros.

- Iniciar o NIDIA de forma automática, ou seja, apenas a informação sobre quais *hosts* devem hospedar o sistema deve servir de entrada para o sistema e ele automaticamente distribui os agentes pelos *hosts*.

Referências Bibliográficas

[Allen et al., 1999] Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., , and E. (1999). State of the practice of intrusion detection technology. Software Engineering Institute: Carnegie Mellon University.

[Balasubramaniyan et al., 1998] Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E. H., and Zamboni, D. (1998). An architecture for intrusion detection using autonomous agents. In *ACSAC*, pages 13–24.

[Bellifemine et al., 2005] Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G. (Data de Acesso: 10/2005). Jade programmers guide. disponível em: http://sharon.cselt.it/projects/jade/doc/programmersguide.pd.

[Campello et al., 2001] Campello, R. S., Weber, R. F., da Silveira Serafim, V., and Ribeiro, V. G. (2001). O sistema de detecção de intrusão asgaard. Florianópolis, SC, Brasil.

[CERT.br, 2005] CERT.br (Data de acesso: 10/2005). Centro de estudos resposta e tratamento de incidentes de segurança no brasil, disponível em: http://www.cert.br.

[Chetan et al., 2005] Chetan, S., Ranganathan, A., and Campbell, R. (2005). Towards fault tolerant pervasive computing. *In IEEE Technology and Society*, 24:38–44.

[CIDF, 2005] CIDF (Data Acesso date: 10/09/2005). Common intrusion detection framework, disponível em: http://www.isi.edu/gost/cidf/.

[da Silva, 2005] da Silva, I. G. L. (Agosto/2005). Projeto e implementação de sistemas multi-agentes: O caso tropos. In *Dissertação de Mestrado*, page 119 páginas, Recife, Brasil.

[de Castro Guerra, 2004] de Castro Guerra, P. A. (2004). Uma abordagem arquitetural para tolerância a falhas em sistemas de software baseados em componentes. In *Tese de Doutorado. UNICAMP. Instituto de Computação Universidade Estadual de Campinas*, Campinas, SP.

[de Lourdes Ferreira, 2003] de Lourdes Ferreira, G. (Novembro, 2003). Um agente inteligente controlador de ações do sistema. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[de Oliveira et al., 2001] de Oliveira, I., Balby, L., and Girardi, R. (2001). Padrões baseados em agentes para a modelagem de usuários e adaptação de sistemas. In *Anais da Quarta Conferência Latino-Americana em Linguagens de Padrões para Programação (SugarLoafPLOP2004)*, pages 265–277, Fortaleza, Ceará, Brasil.

[Dias, 2003] Dias, R. A. (Novembro, 2003). Um modelo de atualização automática do mecanismo de detecção de ataques de rede para sistemas de detecção de intrusão. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[Dias and Nascimento, 2003] Dias, R. A. and Nascimento, E. (2003). Um modelo de atualização automática do mecanismo de detecção de ataques de rede para sistemas de detecção de intrusão. São José dos Campos, Brasil. Proceedings of V Simpósio de Segurança em Informática.

[Fedoruk and Deters, 2002] Fedoruk, A. and Deters, R. (2002). Improving fault-tolerance by replicating agents. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 737–744, New York, NY, USA. ACM Press.

[FIPA, 2005] FIPA (Data de acesso: 01/2005). Fipa, disponível em: http://www.fipa.org/.

[Guangchun et al., 2003] Guangchun, L., Xianliang, L., Jiong, L., and Jun, Z. (2003). Madids: a novel distributed ids based on mobile agent. volume 37, pages 46–53, New York, NY, USA. ACM Press.

[Guerraoui and Schiper, 1997] Guerraoui, R. and Schiper, A. (1997). Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74.

[Guessoum et al., 2002] Guessoum, Z., Briot, J.-P., Charpentier, S., Marin, O., and Sens, P. (2002). A fault-tolerant multi-agent framework. In *AAMAS*, pages 672–673. ACM.

[Guessoum et al., 2005] Guessoum, Z., Faci, N., and Briot, J.-P. (2005). Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform. In *SELMAS '05: Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems*, pages 1–6, New York, NY, USA. ACM Press.

[Haegg, 1997] Haegg, S. (1997). A sentinel approach to fault handling in multi-agent systems. In *Revised Papers from the Second Australian Workshop on Distributed Artificial Intelligence*, pages 181–195, London, UK. Springer-Verlag.

[Hegazy et al., 2003] Hegazy, I. M., Al-Arif, T., Fayed, Z. T., and Faheem, H. M. (2003). A multi-agent based system for intrusion detection. *IEEE Potentials*, 22:28–31.

[IDMEF, 2006] IDMEF, IDXP e CIDF Em busca de uma padronização para Sistemas de Detecção de Intrusão, D. A. M. (Data de acesso: 01/2006). Disponível em: http://www.modulo.com.br/.

[JADE, 2005] JADE (Data de acesso: 10/2005). Jade: Java agent development framework, disponível em: http://jade.tilab.com/.

[Karlsson et al., 2005] Karlsson, B., Bäckström, O., Kulesza, W., and Axelsson, L. (2005). Intelligent sensor networks - an agent-oriented approach. In *Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, Stockholm, Sweden.

[Khan et al., 2005] Khan, Z., Shahid, S., Ahmad, H., Ali, A., and Suguri, H. (2005). Decentralized architecture for fault tolerant multi agent system. In *Autonomous Decentralized Systems, 2005. ISADS 2005*, pages 167– 174. IEEE.

[Klein and Dellarocas, 1999] Klein, M. and Dellarocas, C. (1999). Exception handling in agent systems. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 62–68, New York, NY, USA. ACM Press.

[Kruegel and Toth, 2001a] Kruegel, C. and Toth, T. (2001a). Applying mobile agent technology to intrusion detection. In *ICSE Workshop on Software Engineering and Mobility*, Canada.

[Kruegel and Toth, 2001b] Kruegel, C. and Toth, T. (2001b). Sparta - a mobile agent based intrusion detection system. In *In: IFIP I-NetSec, Kluwer Academic Publishers*, Belgium.

[Kumar and Cohen, 2000] Kumar, S. and Cohen, P. R. (2000). Towards a fault-tolerant multi-agent system architecture. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 459–466, New York, NY, USA. ACM Press.

[Kumar et al., 1999] Kumar, S., Cohen, P. R., and Levesque, H. J. (1999). The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. Technical Report CSE-99-016-CHCC.

[Laprie, 1995] Laprie, J. C. (1995). Dependability: Basic concepts and terminology. *Special Issue of the Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS- 25)*, pages 42–54.

[Laureano, 2004] Laureano, M. A. P. (2004). Uma abordagem para a proteÇÃo de detectores de intrusÃo baseada em mÁquinas virtuais. In *Dissertação de Mestrado. Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná.*, Curitiba, Paraná, Brasil.

[Lemos and Fiadeiro, 2002] Lemos, R. and Fiadeiro, J. L. (2002). An architectural support for self-adaptive software for treating faults. In D. Garlan, J. Kramer, A. W., editor, *Proceedings of the 1st ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02)*, pages 39–42, Charleston, SC, USA.

[Liang et al., 2003] Liang, D., Fang, C.-L., Chen, C., and Lin, F. (2003). Fault tolerant web service. In *APSEC*, pages 310–. IEEE Computer Society.

[Lima, 2002] Lima, C. F. L. (Janeiro, 2002). Proposta de atualização automática dos sistemas de detecção de intrusão por meio de web services. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[Lopes, 2006] Lopes, M. (2006). Sistema de detecção de intrusão remoto. In *Dissertação de Mestrado (em fase de elaboração). Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[Marin et al., 2003] Marin, O., Bertier, M., and Sens, P. (2003). Darx - a framework for the fault tolerant support of agent software.

[Marin et al., 2001] Marin, O., Sens, P., Briot, J.-P., and Guessoum, Z. (2001). Towards adaptive fault-tolerance for distributed multi-agent systems. In *Proceedings of the 3rd. European Research Seminar on Advanced Distributed Systems (ERSADS'2001)*, pages 195–201.

[Mishra, 2001] Mishra, S. (2001). Agent fault tolerance using group communication. In *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Application (PDPTA 2001)*, pages 155–159, Las Vegas, NV.

[Nwana et al., 1999] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). Zeus: a toolkit and approach for building distributed multi-agent systems. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 360–361, New York, NY, USA. ACM Press.

[Oliveira, 2005] Oliveira, A. A. P. (2005). Sociedade de agentes para a monitoração de ataques e respostas automatizadas. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[Oliveira et al., 2005] Oliveira, A. A. P., Abdelouahab, Z., and Nascimento, E. (2005). Using honeypots and intelligent agents in security incident responses and investigation of suspicious actions in interconnected computer systems. In *Proceedings of the E-Crime And Computer Evidence Conference Mônacog*, Monaco.

[Oliveira and Abdelouahab, 2006] Oliveira, E. J. S. and Abdelouahab, Z. (2006). Secure agent communication languages with xml security standards.

[Pestana, 2005] Pestana, F. A. (2005). Proposta de atualização automática dos sistemas de detecção de intrusão por meio de web services. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil.

[Ramachandran and Hart, 2004] Ramachandran, G. and Hart, D. (2004). A p2p intrusion detection system based on mobile agents. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 185–190, New York, NY, USA. ACM Press.

[Saidane et al., 2003] Saidane, A., Deswarte, Y., and Nicomette, V. (2003). An intrusion tolerant architecture for dynamic content internet servers. In *SSRS '03: Proceedings of*

the 2003 ACM workshop on Survivable and self-regenerative systems, pages 110–114, New York, NY, USA. ACM Press.

[Santos and Nascimento, 2003] Santos, G. L. F. and Nascimento, E. (2003). An automated response approach for intrusion detection security enhancement. Marina Del Rey, California, USA.

[Santos and Campello, 2001] Santos, O. M. and Campello, R. S. (2001). Estudo e implementação de mecanismos de tolerância a falhas em sistemas de detecção de intrusão. Santa Maria, UNIFRA.

[Security, 2005] Security, M. (Data de acesso: 10/2005). Módulo security, disponível em: http://www.modulo.com.br.

[Serugendo and Romanovsky, 2002] Serugendo, G. D. M. and Romanovsky, A. B. (2002). Designing fault-tolerant mobile systems. In Guelfi, N., Astesiano, E., and Reggio, G., editors, *FIDJI*, volume 2604 of *Lecture Notes in Computer Science*, pages 185–201. Springer.

[Siqueira and Abdelouahab, 2006] Siqueira, L. and Abdelouahab, Z. (2006). A fault tolerance mechanism for network intrusion detection system based on intelligent agents (nidia). In *3rd Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2006)*, Monaco.

[Townend and Xu, 2003] Townend, P. and Xu, J. (2003). Fault tolerance within a grid environment. *In Proceedings of AHM2003*, page 272.

[UML, 2005] UML (Data de acesso: 24/10/2005). The unified modeling language, disponível em: http://www.uml.org/.

[Vuong and Fu, 2001] Vuong, S. T. and Fu, P. (2001). A security architecture and design for mobile intelligent agent systems. *SIGAPP Appl. Comput. Rev.*, 9(3):21–30.

[Wallace, 2005] Wallace, S. A. (2005). S-assess: a library for behavioral self-assessment. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 256–263, New York, NY, USA. ACM Press.

[Wasniowski, 2005] Wasniowski, R. A. (2005). Multi-sensor agent-based intrusion detection system. In *InfoSecCD '05: Proceedings of the 2nd annual conference on*

Information security curriculum development, pages 100–103, New York, NY, USA. ACM Press.

[Yepes, 2005] Yepes, I. (Data de acesso: 08/2005). Disponível em: http://www.geocities.com/igoryepes/agentes.htm.

[Zhang, 2005] Zhang, Y. (2005). Towards fault tolerance for multiagent systems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1387–1387, New York, NY, USA. ACM Press.

[Zhicai et al., 2004] Zhicai, S., Zhenzhou, J., and Mingzeng, H. (2004). A novel distributed intrusion detection model based on mobile agent. In *InfoSecu '04: Proceedings of the 3rd international conference on Information security*, pages 155–159, New York, NY, USA. ACM Press.

[Zorzo and Meneguzzi, 2005] Zorzo, A. F. and Meneguzzi, F. R. (2005). An agent model for fault-tolerant systems. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 60–65, New York, NY, USA. ACM Press.