UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA
DE ELETRICIDADE

GERENCIAMENTO E INTEGRAÇÃO DAS BASES DE DADOS DE SISTEMAS DE DETECÇÃO DE INTRUSÕES

EMANOEL COSTA CLAUDINO SILVA

São Luís
2006

EMANOEL COSTA CLAUDINO SILVA

GERENCIAMENTO E INTEGRAÇÃO DAS BASES DE DADOS DE SISTEMAS DE DETECÇÃO DE INTRUSÕES

Dissertação de Mestrado submetida à Coordenação do Programa 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, área de concentração: Ciência da Computação.

Orientador: Prof. Ph.D. Zair Abdelouahab

São Luís
2006

Silva, Emanoel Costa Claudino.

Gerenciamento e Integração das Bases de Dados de Sistemas de Detecção de Intrusões/Emanoel Costa Claudino Silva – São Luís, 2007

149 f.

Impresso por computador (fotocópia).

Orientador: Zair Abdelouahab.

Dissertação (Mestrado) - Universidade Federal do Maranhão, Programa de Pós-graduação em Engenharia de Eletricidade, 2007.

1. Redes (Computadores) – Segurança. 2. Sistemas Multiagentes. 3. Segurança de Redes. I Abdelouahab, Zair. II. Título.

CDU 004.7.056

GERENCIAMENTO E INTEGRAÇÃO DAS BASES DE DADOS DE SISTEMAS DE DETECÇÃO DE INTRUSÕES

Emanoel Costa Claudino Silva

Dissertação aprovada em 19 de dezembro de 2006.

Prof. Zair Abdelouahab, Ph. D.
(Orientador)

Prof. Omar Andres Carmona Cortes, Dr.
(Membro da Banca Examinadora)

Prof. Mario Antonio Meireles Teixeira, Dr.
(Membro da Banca Examinadora)

Denivaldo Cícero Pavão Lopes, Dr.
(Membro da Banca Examinadora)

Ao meu pai, Antonio Claudino.
Aos meus filhos, Emanoel e Mariana.
À Arabela, minha esposa.

“Graças, porém, a Deus que em Cristo sempre nos conduz em triunfo, e, por meio de nós, manifesta em todo lugar a fragrância do seu conhecimento.”

São Paulo

AGRADECIMENTOS

Inicialmente, agradeço a Deus por todas as provisões que tornaram este dia uma realidade.

À minha família pela compreensão nas minhas ausências em nossas reuniões familiares, principalmente minha esposa e filhos que compartilharam comigo todos os momentos de “altos e baixos” durante esses anos.

À Coordenação do Programa de Pós-Graduação de Engenharia Elétrica por ter-me concedido à oportunidade de realizar este Mestrado.

Ao Professor Zair Abdelouahab pela orientação e paciência durante a realização deste trabalho.

Ao Professor Mario Meireles pelo apoio, e constante incentivo.

Aos meus companheiros de mestrado, Emerson e Bysmarck.

A todos aqueles que, de forma direta ou indireta contribuíram para a realização desta dissertação.

RESUMO

A Segurança digital tem se tornado fator inegociável para instituições de diversos domínios. Os Sistemas de Detecção de Intrusão (SDIs) têm surgido como uma solução para detecção e correção de intrusão de forma pró-ativa. Assim, vários modelos de SDIs têm surgido para, identificando, reportando e respondendo a estes incidentes, diminuir a probabilidade de comprometimento dos sistemas computacionais ligados em rede. Diante desta diversidade de soluções, faltam propostas de padronização das informações utilizadas por estes Sistemas, bem como de mecanismos de interoperabilidade e troca de informações entre as soluções em uso. Esta dissertação, propõem um modelo, uma arquitetura e uma implementação de um Gerenciador de Informações para SDIs, usando as tecnologias de Sistemas Multiagentes e *Web Services.* O objetivo do Gerenciador de Informações é manter de forma segura e atualizada as informações que são necessárias ao desenvolvimento das funções inerentes a um SDI. É proposto também, um padrão de formato para armazenamento desses dados, de forma a inserir no ambiente requisitos como: Armazenamento Unificado, Acesso Transparente, Geração de Dados Uniforme e Facilidade de Interoperabilidade.

Palavras-chaves: Gerenciamento de Informações, Sistemas Multiagentes, Sistemas de Integração de Dados para SDIs, Segurança de Redes.

ABSTRACT

The digital security has become an important factor for institutions of diverse domains. The Intrusion Detection Systems (IDS) have appeared as a solution for detention and correction of intrusion of pro-active way. Thus, some models of SDIs have appeared to diminish the probability of compromising of on computational systems connected in net, identifying, reporting and answering to these incidents. In face to that diversity of solutions, they lack proposals of standardization of the information used for these Systems, as well as of mechanisms of interoperability and exchange of information between the solutions in use. This dissertation, proposes a model, an architecture and an implementation of a SDI´s Information Manager, using the technologies of Multi-Agents Systems and Web Services. The objective of the Information Manager is to keep the information that are necessary to the development of the inherent functions of a SDI, in a safe and updated way. We also propose a standard of format for storage of these data to insert requirements in the environment, as: Unified Storage, Transparent Access, Uniform Generation of Data and Friendly Interaction.

Index Terms: Information Manager, Intrusion Detection Systems, Multi-Agents Systems, Data Integration Systems for IDS, Net Security.

LISTA DE ABREVIATURAS DE SÍMBOLOS

ACL	Agent Communication Language
API	Application Programming Interface
CAP	Common Alerting Protocol
CERT.br	Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil
CIDF	Common Intrusion Detection Framework
CSIRT	Computer Security Incident Response Team
CVE	Common Vulnerabilities and Exposures
DFDB	Data Forense Data Base
DOM	Document Object Model
DoS	Denial of Service
FIPA	Foundation for Intelligent Physical Agents
IDE	Integrated Development Environment
IDMEF	Intrusion Detection Message Exchange Format
IETF	Internet Engineering Task Force
IIDB	Intrusion Information Data Base
IODEF	Incident Object Description and Exchange Format
IP	Internet Protocol
JADE	Java Agent DEvelopment Framework
JAM	Java Agents for Meta-learning over Distributed Databases
JDBC	Java Database Connect
JDK	Java Developer Kit
LGPL	Lesser General Public License
MAS	Multi-Agents Systems
MCA	Main Controller Agent
MDA	Model Driven Architecture
NIDIA	Network Intrusion Detection System Based on Intelligent Agent
NIDS	Network Intrusion Detection Systems
NSM	Network Security Monitor
OKDB	Optimized KeyWord Data Base
PASSI	Process for Agent Societies Specification and Implementation
RADB	Response Action Data Base
RMI	Remote Method Invocation
SAX	Simple API for XML

SCA	System Controller Agent
SDI	Sistemas de Detecção de Intrusão
SEA	Security Evaluation Agent
SGBD	Sistema Gerenciador de Banco de Dados
SGML	Standard Generalized Markup Language
SMA	System Monitoring Agent
SOAP	Simple Object Access Protocol
STDB	Strategy Data Base
SUA	System Updating Agent
TI	Tecnologia da Informação
UCP	Unidade Central de Processamento
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
XML	Extensible Markup Language
WSDL	Web Services Description Language

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Na Internet, os invasores de redes são navegadores sofisticados. Eles vêm de fora da empresa através da Internet, alteram páginas da *Web*, lançam ataques de DoS (*Denial of Service* – negação de serviço), acessam e alteram informações corporativas etc. Eles também podem partir de dentro da rede, iniciando violações sofisticadas que contornam ou atravessam *firewalls*, transmitindo informações confidenciais ou modificando ilegalmente privilégios. Como fatores complicadores podem-se citar: um crescimento contínuo de vulnerabilidades, condições geopolíticas incertas e restrições de orçamento para TI (Tecnologia da Informação).

Os sistemas responsáveis por auxiliar na segurança exigem vigilância constante e monitoramento pró-ativo para serem eficazes. Os alertas e eventos de segurança precisam ser verificados e analisados 24 horas por dia, 7 dias por semana, para diferenciar as ameaças reais à segurança dos falsos alarmes. Entretanto, o tempo e o esforço associados à verificação de eventos de segurança pode ser um fardo pesado para qualquer organização.

A criação e a implementação de uma arquitetura de segurança corporativa que inclua uma infra-estrutura de detecção de intrusão pró-ativa é um importante passo para garantir a segurança dos bens mais importantes de uma organização. Embora muitas organizações implementem *firewalls* como o principal recurso de vigilância para impedir acesso não autorizado, a proteção robusta de redes e servidores somente pode ser garantida quando uma defesa em camadas for empregada [64].

Os SDIs (Sistemas de Detecção de Intrusão) são utilizados como parte da solução para o problema de segurança das redes de computadores.

Eles são compostos por *software* e/ou *hardware* que automatizam o processo de monitoramento de eventos que ocorrem em um sistema computacional ou rede, com o objetivo de identificar ameaças e responder a essas ações intrusivas [7].

1.1 Definição do Problema

Diariamente são descobertas novas vulnerabilidades nos sistemas existentes e são relatados novos casos de ataques bem sucedidos contra diversos sítios. No ano de 2004 foram feitas 3.780 (três mil, setecentos e oitenta) notificações de vulnerabilidades ao CERT®/CCI; e no primeiro semestre de 2005 foram 2.874 (duas mil, oitocentos e setenta e quatro) notificações [12]. Somente no primeiro semestre de 2006, o CERT.br (Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil) registrou 59.576 relatos de incidentes de segurança nos mais diversos níveis [13].

Com o aumento do número de invasões a redes de computadores, vários modelos de SDIs [4,8,27,36,49,59] têm surgido para identificar, reportar e responder a estes incidentes de forma automática, e diminuir a probabilidade de comprometimento dos sistemas computacionais ligados em rede.

Diante desta diversidade de soluções, faltam propostas de padronização das informações utilizadas por SDIs, bem como dos mecanismos de interoperabilidade e troca de informações entre as soluções em uso.

Outra importante observação que deve ser feita, é que um SDI toma decisões com base em um conjunto de informações pré-existentes. Para que este execute suas funções de forma eficiente se faz necessário que estas informações estejam atualizadas. Esta atualização é proporcionada através da

obtenção de informações geradas internamente e externamente ao SDI, e assim, para que este processo de troca de dados seja otimizado, é imprescindível que esses dados obedeçam a um formato padrão.

Devido à inexistência de um padrão definido para representar esses dados relativos à segurança, e tampouco de como o compartilhamento dessas informações deve ser feito, apresenta-se neste trabalho uma proposta para a formatação e armazenamento das informações necessárias ao desenvolvimento das funções inerentes a um SDI, assim como uma proposta de arquitetura baseada em Sistemas Multiagentes e *Web Services* para manter seguras, atualizadas e permitir disponibilização automática dessas informações a outros SDIs como também a outras entidades externas, como por exemplo, os CSIRT (*Computer Security Incident Response Team)* [5,12].

1.2 Objetivos Geral e Específico

O objetivo desta dissertação de mestrado é desenvolver um modelo que proporcione o compartilhamento e o gerenciamento do conjunto de informações utilizadas por um SDI. Para tanto, neste trabalho é proposto o Módulo Gerenciador de Informações [15,16] que tem por objetivo manter seguras e atualizadas as informações necessárias ao desenvolvimento das funções de um SDI, utilizando-se as tecnologias de Sistemas Multiagentes e *Web Services*.

São objetivos específicos:

- Identificar as informações necessárias para que um Sistema de Detecção de Intrusão execute suas tarefas de forma eficiente;
- Apresentar uma proposta de formato de dados para as informações identificadas utilizando-se XML (*Extensible Markup*

Language), proporcionando um compartilhamento otimizado de informações entre SDIs, e entre SDIs e outras entidades externas;

- Integrar essas informações em repositórios, objetivando aumentar a disponibilidade das informações, diminuição no tempo gasto para acesso e melhorar a capacidade de atualização das mesmas;
- Propor um modelo que forneça acesso transparente a esses repositórios e permita a interoperabilidade entre Sistemas de Detecção de Intrusão;
- Desenvolver protótipo do modelo proposto;
- Aplicar o modelo proposto ao SDI NIDIA (*Network Intrusion Detection System based on Intelligent Agent*).

1.3 Justificativa e Relevância

Dentro de qualquer organização, as decisões são tomadas com base em informações, assim quanto mais precisas às informações mais acertadas são as decisões. Dentro de um Sistema de Detecção de Intrusão não é diferente, pois a detecção de ataques e as respostas adequadas a essas ameaças baseiam-se em informações que devem está disponíveis e atualizadas considerando-se que freqüentemente são encontradas novas formas de ataques ou a exploração de novas vulnerabilidades nos sistemas existentes [5,12,13,36].

O aumento da insegurança no que se refere às redes de computadores levou a existência de vários modelos de Sistemas de Detecção de Intrusão, gerando vários modelos de informações. Essa heterogeneidade de

modelos dificulta a interação entre os diversos SDIs e a troca de informações. Assim a adoção de um modelo que proporcione a internacionalização dessas informações criaria um ambiente propício à comunicação entre os diversos tipos de SDIs possibilitando a troca e atualização constante de informações. Essa capacidade de interação pode ser usada para avaliar o nível de segurança do domínio, a eficiência das ferramentas usadas para prover esta segurança e a eficiência do próprio SDI. Comparar informações geradas por vários SDIs e analisar tipos de ataques sofridos, ações de respostas executas, estratégias de ações adotadas com certeza resultará em um refinamento das ações de segurança adotadas pelos administradores, proporcionando ambientes mais seguros.

Além do compartilhamento de informações entre SDIs, é importante lembrar que um SDI é um sistema complexo, composto por vários subsistemas especializados que executam tarefas específicas e que geram e necessitam constantemente de informações[35]. Nesse contexto, torna-se imprescindível que todos os componentes de um SDI gerem informações no mesmo formato facilitando o acesso e o gerenciamento desses dados.

1.4 Organização do Trabalho

Esta dissertação encontra-se organizada em sete Capítulos. No primeiro Capítulo, é apresentado o cenário atual em nível de segurança digital em que se encontram as empresas e instituições. Dentro deste cenário, destaca-se o uso de SDIs como ferramenta de segurança e a necessidade de informações atualizadas para ação eficaz desse tipo de ferramenta. Apresenta-se também os objetivos gerais e específicos, bem como a organização da dissertação.

No Capítulo 2, faz-se uma introdução sobre as tecnologias usadas na solução do problema exposto. Mostra-se ainda, uma visão geral dos Sistemas de Detecção de Intrusão, enfatizando-se as vantagens desses sistemas baseados na tecnologia de agentes, e em especial apresenta-se o SDI NIDIA (*Network Intrusion Detection System based on Intelligent Agents*).

No Capítulo 3, descrevem-se as informações necessárias para que um SDI execute suas tarefas de forma eficiente, sendo proposto um modelo para a padronização destas informações, baseado em XML.

No Capítulo 4, propõem-se o Módulo Gerenciador de Informações, ou seja, um sistema baseado na tecnologia de agentes e *Web Service.* A sua arquitetura é apresentada através de um modelo em camadas e sua funcionalidade é descrita a partir do detalhamento dos agentes que compõem cada uma de suas camadas.

No Capítulo 5, relata-se o processo de construção do Módulo Gerenciador de Informações, através do uso de ferramentas como PASSI (*Process for Agent Societies Specification and Implementation*), JADE (*Java Agent DEvelopment Framework*) e Xindice. Além disso, detalha-se a implementação dos agentes, destacando-se os agentes *SeacherAgent* e *ReceiverAgent*, responsáveis pela extração e armazenamento de informações.

No Capítulo 6, mostram-se as simulações realizadas com o grupo de agentes do Gerenciador de Informações interagindo com um protótipo do NIDIA e realizando as tarefas de disponibilização e armazenamento de informações.

O Capítulo 7 dedica-se às conclusões finais, ressaltando-se as contribuições desta pesquisa, indicando algumas sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capitulo, alguns conceitos chaves sobre Sistemas de Detecção de Intrusão (SDIs) e trocas de mensagens de segurança serão apresentados. Apresenta-se também o *Network Intrusion Detection System Based on Intelligent Agent* (NIDIA) [38] que é um SDI baseado em agentes inteligentes o qual será usado para integrar o Gerenciador de Informações [15,16] proposto nesta dissertação. Em seguida descrevem-se as tecnologias utilizadas como base neste trabalho.

2.1 Sistemas de Detecção de Intrusão

O conceito de SDI foi proposto pela primeira vez em 1980 por James Anderson [1], sendo difundida somente em 1987, com a publicação do artigo "*An Intrusion Detection Model*" (Um Modelo de Detecção de Intrusão) por Dorothy Denning [20]. Em 1988, pelo menos três propostas de SDI foram apresentadas: "*NIDX: An Expert System for Real-Time Network Intrusion Detection*" (NIDX: Um Sistema Especialista para Detecção de Intrusos em Redes de Computadores em Tempo Real) [9]; "*A Expert systems in intrusion detection: A case study*" (Sistemas Especialistas em Detecção de Intrusos: Um Estudo de Caso) [51]; e "*Haystack: An Intrusion Detection System*" (Haystack: Um Sistema de Detecção de Intrusão) [56]. Nos anos seguintes, vários protótipos foram desenvolvidos.

Sundaram [60], Jackson [30], Anderson [2], Bace [6], Bishop [10], Eskin [23,24] e outros [8,25,37] pesquisadores têm desenvolvido várias pesquisas na área de Sistemas de Detecção de Intrusão, modelos e métodos. Estes esforços incluem os seguintes campos:

- Modelos Genéricos de Detecção de Intrusão;

- Modelo NSM (*Network Security Monitor*);
- Modelo baseado em Agentes Autônomos;
- Modelo de Detecção de Intrusão Baseado em Comportamento;
- Modelos Gerados a partir de Padrões de Predicativos;
- Modelo de Detecção de Intrusão Baseado em Conhecimento.

2.1.1 Funções de um SDI

Segundo Crosbie e Spafford [18], uma intrusão pode ser definida como sendo um conjunto de ações que tentam comprometer a integridade, a confidencialidade e/ou a disponibilidade de recursos em um sistema computacional.

Um SDI realiza o processo de monitorar os eventos ocorridos em um sistema ou rede de computadores, analisando-os através de assinaturas de intrusão ou desvio de padrão em perfis de comportamento. As intrusões são causadas por usuários não autorizados tentando acessar um sistema através da Internet e/ou por usuários legítimos tentando abusar de seus privilégios, motivados por ganhos financeiros, vingança, necessidade de aceitação ou respeito, idealismo, curiosidade ou busca de emoção, aprendizado e/ou espionagem industrial.

Assim, a principal tarefa a ser desempenhada por um SDI é defender um sistema computacional pela detecção de um ataque e desencadeamento automático de uma ou várias ações em contra-ataque. Para tanto realiza três tarefas padrões [52]:

- Monitoração do sistema (*Monitoring*);
- Notificação aos ataques (*Notification*);
- Tomada das contramedidas necessárias (*Response*).

As três tarefas padrões, desenvolvidas por um SDI, anteriormente listadas são ilustradas na Figura 2.1.

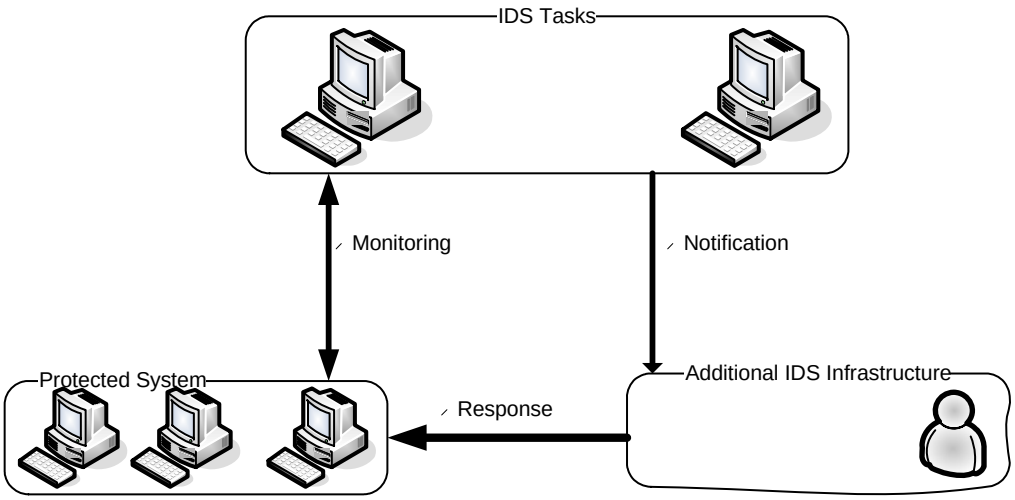

Figura 2.1. Tarefas Padrões de um SDI.

2.1.2 Modelo Conceitual de um SDI

Devido a grande variedade de SDIs, um modelo chamado CIDF (*Common Intrusion Detection Framework*) [14,58] foi proposto. Este modelo agrupa um conjunto de componentes que define uma ferramenta do tipo SDI.

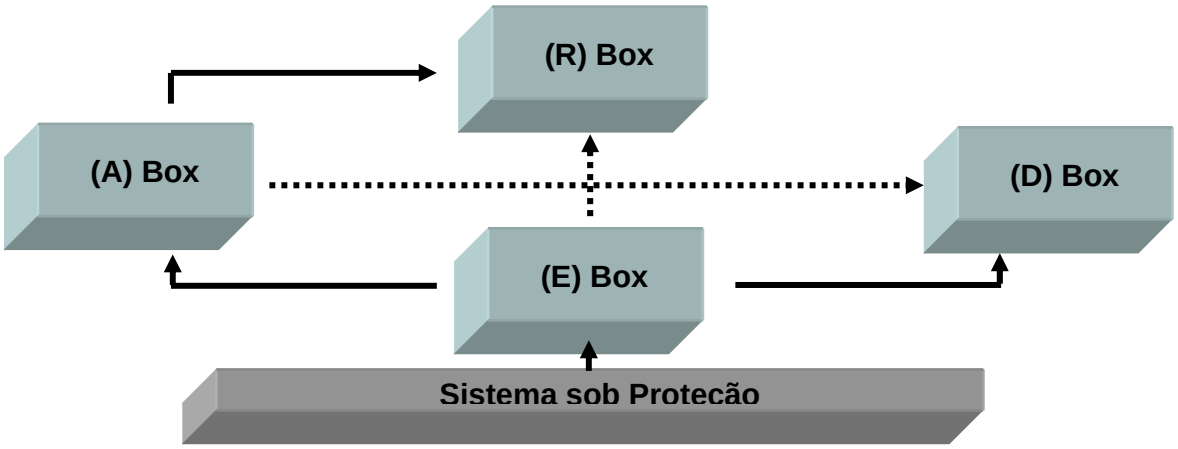

Figura 2.2. Modelo CIDF.

A Figura 2.2 apresenta o Modelo CIDF e os relacionamentos entre seus componentes. A seguir descreve-se a função de cada componente do modelo conceitual:

- **Gerador de Eventos (E-box)**: a função deste componente é obter os eventos a partir do meio externo ao CIDF, ou seja, ele "produz" os eventos, mas não os processa, isso fica a cargo do componente especializado na função de processamento, o A-box, que, por sua vez, após analisar os eventos (por exemplo, violação de política, anomalias e intrusão) envia os resultados para outros componentes;
- **Analisador de Eventos (A-box)**: este componente, basicamente, recebe as informações de outros componentes, as analisa e as envia de forma resumida para outros componentes, ou seja, recebe os dados de forma bruta, faz um refinamento e os envia para outros;
- **Base de Dados de Eventos (D-box):** a função deste componente é armazenar os eventos e/ou resultados para uma necessidade futura;
- **Unidade de Resposta (R-box)**: este componente é responsável pelas ações, ou seja, pelas contramedidas tais como, matar o processo, reinicializar a conexão, alterar a permissão de arquivos e notificar as estações de gerência.

Algumas características desejáveis destes componentes são:

- Reutilização: SDIs podem ser reutilizados em um contexto diferente do qual foram originalmente desenvolvidos, ou seja, devem ser configuráveis de forma a adaptarem-se a ambientes distintos;

- Modulares: eles são elaborados em módulos com funções distintas;
- Compartilhamento de Informações: SDIs podem compartilhar/trocar informações entre si, via API ou através da rede para uma melhor precisão na identificação de ataques;
- Auto-Adaptação: componentes novos devem, automaticamente, identificar os demais componentes;
- Composição: O grupo de componentes pode atuar mutuamente de forma a dar a impressão de ser um único elemento.

2.1.3 Formas de implementação e integração de SDIs

Nesta subseção descreve-se as formas de implementação de um SDI, ou seja, SDIs baseados em *host*, SDIs baseados em rede e SDIs híbridos.

2.1.3.1 SDIs baseados em *host*

Examina trilhas de auditoria e arquivos de logs para monitorar eventos locais em um sistema de computador em particular, podendo detectar ataques realizados através de programas nocivos residentes em um *host*.

Zirkle [69] descreve um SDI baseado em host como um software carregado como parte do sistema a ser monitorado. Este software executa as seguintes tarefas:

- Usam arquivos de log e/ou agentes que auditam o sistema como fonte de dados;
- Monitoram o tráfego de comunicação tanto de entrada quanto de saída com outros computadores;

- Checam a integridade do sistema de arquivos e vigiam processos suspeitos, incluindo mudança dos arquivos de sistemas e privilégios de usuários.

Software de detecção baseado em *host* é particularmente eficaz em detectar ataques *trusted-insider* (ativado por anomalias). Porém, um inconveniente para detecção de intrusão baseada em *host* é que o software de monitoramento deve ser instalado em cada computador da rede a ser protegida.

2.1.3.2 SDIs baseado em rede

A maioria dos Sistemas de Detecção de Intrusão está classificada nessa categoria. Esses sistemas detectam intrusos através da captura e análise de pacotes que trafegam na rede, podendo, com isso, monitorar vários *hosts* conectados ao mesmo segmento de rede.

Northcutt [42,45] descreve Sistemas de Detecção de Intrusão Baseado em Rede ou NIDS (*Network Intrusion Detection Systems*) como um sistema que monitora o tráfego em um segmento de rede, usando esse tráfego como origem de dados. A implementação de NIDS requer:

- Uma interface com a rede, que é colocada numa modalidade para capturar todo o tráfego da rede que cruza seu segmento da rede;
- Sensores que monitorem os pacotes que trafegam nesse segmento de rede devem estar disponíveis.

O objetivo é determinar se o fluxo do pacote combina uma assinatura conhecida. Há três assinaturas que são particularmente importantes:

- Assinaturas em *strings*, que procuram por um texto que indica um possível ataque;

- Assinaturas em porta, que executam o monitoramento simples de tentativas de conexões em portas que são usadas freqüentemente para ataques, e;
- Assinatura em cabeçalho, que procuram por ataques ou combinações ilógicas em cabeçalhos de pacotes.

2.1.3.3 SDIs híbridos

Modelo que tenta agrupar o que há de melhor nos modelos baseado em *host* e baseado em rede. A seguir lista-se as principais vantagens que o modelo híbrido tentar agrupar.

Os SDIs baseados em rede possuem as seguintes características:

- Localizados em pontos estratégicos, podem monitorar uma rede bem extensa;
- Tem pouco impacto na rede monitorada, exigindo um mínimo esforço de instalação, pois os sensores são dispositivos passivos;
- Diversas técnicas podem ser usadas para camuflar os sensores de rede, deixando-os invisíveis aos intrusos;
- É possível monitorar ataques distribuídos à rede.

Dentre as vantagens da abordagem baseada em *host* destacamos:

- Monitorar o acesso à informação em termos de "quem acessou o quê";
- Mapear atividades problemáticas com um usuário específico;
- Rastrear mudanças de comportamento associadas com mau uso;
- Operar em ambientes criptografados;

- Distribuir a carga do monitoramento ao redor dos diversos computadores da rede.

2.1.4 Métodos de Análise de Detecção de Intrusão

Para determinar a ocorrência de uma intrusão usando SDIs, existem dois métodos principais de análise e detecção de intrusão [7], que são a detecção por anomalia e a detecção por abuso.

2.1.4.1 Detecção por anomalia

Os sistemas de detecção de intrusão por anomalia baseiam-se na premissa de que um ataque difere da atividade normal, podendo dessa forma ser identificado.

O método de detecção de intrusos por anomalia é baseado na observação de desvios de comportamentos esperados em atividades relevantes dos usuários ou processos do sistema monitorado. Os detectores de anomalia constroem perfis que representam o comportamento normal dos usuários, *hosts* ou conexões de rede. Esses perfis são gerados através de dados históricos coletados durante um período normal de operação.

Porém, nem sempre a atividade anômala corresponde a uma atividade intrusiva. Por exemplo, um usuário que sempre trabalhou com determinada carga de uso da UCP pode necessitar de mais poder de processamento e nem por isso ele é um intruso.

2.1.4.2 Detecção por abuso

Os detectores por abuso analisam a atividade do sistema observando os eventos ou um conjunto de eventos que sejam semelhantes a um padrão pré-determinado que descreva uma intrusão conhecida.

Este tipo de detecção está relacionado à identificação de comportamentos intrusivos correspondentes à exploração de vulnerabilidades conhecidas, comumente chamadas de assinaturas de ataque. As assinaturas não servem apenas para detectar intrusões consumadas, elas são úteis também para identificar tentativas de ataque. Assim, quando um conjunto de eventos satisfaz apenas parcialmente uma assinatura, pode ser um indicativo de uma intrusão que está acontecendo.

2.1.5 Vantagens na utilização de um SDI

Segundo Bace e Mell [7], pode-se listar como principais vantagens na utilização de SDIs:

- **Detecção de ataques explorando vulnerabilidades que não podem ser corrigidas**: através de vulnerabilidades conhecidas, que em alguns casos não podem ser corrigidas, o invasor pode penetrar em muitas redes. A exemplo disso, têm-se os sistemas herdados, onde o sistema operacional não pode ser atualizado. Mesmo em sistemas passíveis de atualização, os administradores podem não dispor de tempo para instalar todos os *patches* necessários em uma grande quantidade de computadores. Outra situação ocorre quando o próprio negócio requer serviços e protocolos de rede que são notoriamente problemáticos e sujeitos a ataques. Embora, de maneira ideal, todas essas vulnerabilidades possam ser sanadas, na prática isso nem sempre é possível. Assim, uma boa solução pode ser o uso de SDI para detectar e, possivelmente, prevenir tentativas de ataque explorando estas vulnerabilidades;

- **Prevenir que intrusos examinem a rede**: quando os intrusos atacam uma rede, eles geralmente o fazem em etapas. A primeira etapa é examinar o sistema em busca de alguma vulnerabilidade. Na ausência de um SDI, o atacante se sente livre para fazer o seu trabalho sem riscos e pode descobrir com facilidade os pontos fracos do sistema e explorá-los. Com um SDI, embora o iminente intruso possa continuar a analisar o sistema, ele será detectado automaticamente e os devidos alertas serão dados;
- **Documentação de ameaça de invasão**: é importante entender a freqüência e as características dos ataques em um sistema computacional para medir o grau de hostilidade de um ambiente. Com isso, é possível tomar as medidas de segurança adequadas para proteger a rede. Um SDI pode quantificar, caracterizar e verificar as ameaças de intrusão e mau uso, interna ou externamente;
- **Controle de qualidade**: com o uso contínuo de um SDI, padrões de utilização do sistema monitorado e problemas detectados tornam-se evidentes. Com essas informações é possível tornar visíveis falhas no projeto, configuração e gerenciamento da segurança, proporcionado ao administrador à possibilidade de corrigi-las antes de ocorrer um incidente.

Essas vantagens sofrem variações de acordo com alguns aspectos básicos que devem ser considerados quando do desenvolvimento de um SDI, tais como: os métodos de análises adotados, a escolha do tipo de monitoramento, o tempo entre a coleta e análise dos dados e o tipo de resposta a ser dada.

2.2 Formatos para troca de Mensagens de Segurança

A falta de um padrão para representar e trocar dados de segurança é um grave empecilho para o compartilhamento dessas informações. Aqui são apresentados alguns formatos de dados que foram propostos para troca de informações de segurança dos computadores.

2.2.1 Intrusion Detection Message Exchange Format (IDMEF)

Projetado pela IETF (*Internet Engineering Task Force*), o IDMEF (*Intrusion Detection Message Exchange Format*) ainda é um trabalho em andamento, e possui o propósito de definir formatos de dados e procedimentos para o compartilhamento de informações entre Sistemas de Detecção de Intrusão e sistemas de gerenciamento. Esses sistemas de gerenciamento são Grupos de Resposta a Incidentes de Segurança em Computadores ou *Computer Security Incident Response Team (*CSIRT), que podem necessitar interagir com SDIs [28].

A proposta de implementação do IDMEF permite o desenvolvimento de uma linguagem capaz de descrever alertas de detecção de intrusão. Por meio do IDMEF, pode-se obter uma base de dados que contenha dados advindos de diferentes SDIs tornando possível o acompanhamento de incidentes que ocorram entre vários domínios. Ele visa facilitar a tarefa de correlacionar alerta de segurança em busca de padrões de ataques.

2.2.2 Incident Object Description and Exchange Format (IODEF)

O propósito do IODEF (*Incident Object Description and Exchange Format*) [29] é o de definir uma representação de dados, provendo um *framework*, para o compartilhamento de informações referentes a incidentes de

segurança, comumente trocadas entre grupos de resposta a incidentes de segurança em computadores.

O IODEF objetiva ampliar e melhorar a capacidade operacional dos CSIRTs. A adoção pela comunidade do IODEF poderá fornecer uma maior habilidade para resolver incidentes de segurança por meio da colaboração simplificada e do compartilhamento de dados. O formato estruturado estipulado pelo IODEF visa permitir:

- Uma automação crescente no processamento de dados de incidentes de segurança;
- Menor esforço em normalizar dados similares procedentes de diferentes fontes;
- Um formato comum a partir do qual poderão ser construídas ferramentas interoperáveis para a manipulação de incidentes de segurança.

O IODEF, assim como o IDMEF, ainda é um trabalho em desenvolvimento pela IETF. Estes dois formatos são complementares.

2.2.3 Common Alerting Protocol (CAP)

O CAP (*Common Alerting Protocol*) [11] é um formato simples e abrangente para a troca de todos os tipos de alertas de emergência. Esses alertas de emergência têm por finalidade informar ao público sobre possíveis perigos ou ameaças que possam causar algum tipo de dano. Este é um formato aberto e não proprietário de mensagens digitais para todos os tipos de alertas e notificações, sendo compatível com *Web Services.*

A função principal das mensagens de alerta no formato CAP é prover uma única mensagem que tenha a capacidade de ativar todos os tipos

de sistemas de alerta. Uma função secundária, é padronizar os alertas emitidos por diversas fontes de tal forma que eles possam ser agregados e comparados, auxiliando na detecção de padrões. Assim, o CAP, objetiva facilitar a detecção de padrões emergentes nos diversos alertas emitidos ao público que podem indicar uma ameaça não detectada ou atos hostis.

2.2.4 Comparativo entre os formatos apresentados

As propostas do IETF, IDMEF e IODEF, são formatos de dados para alertas possivelmente gerados por um sistema automatizado de detecção de intrusão. Esses alertas têm como objetivo informar ao administrador de rede sobre possíveis incidentes de segurança e também fornecê-lhe dados a serem enviados a um Grupo de Resposta a Incidentes de Segurança em Computadores.

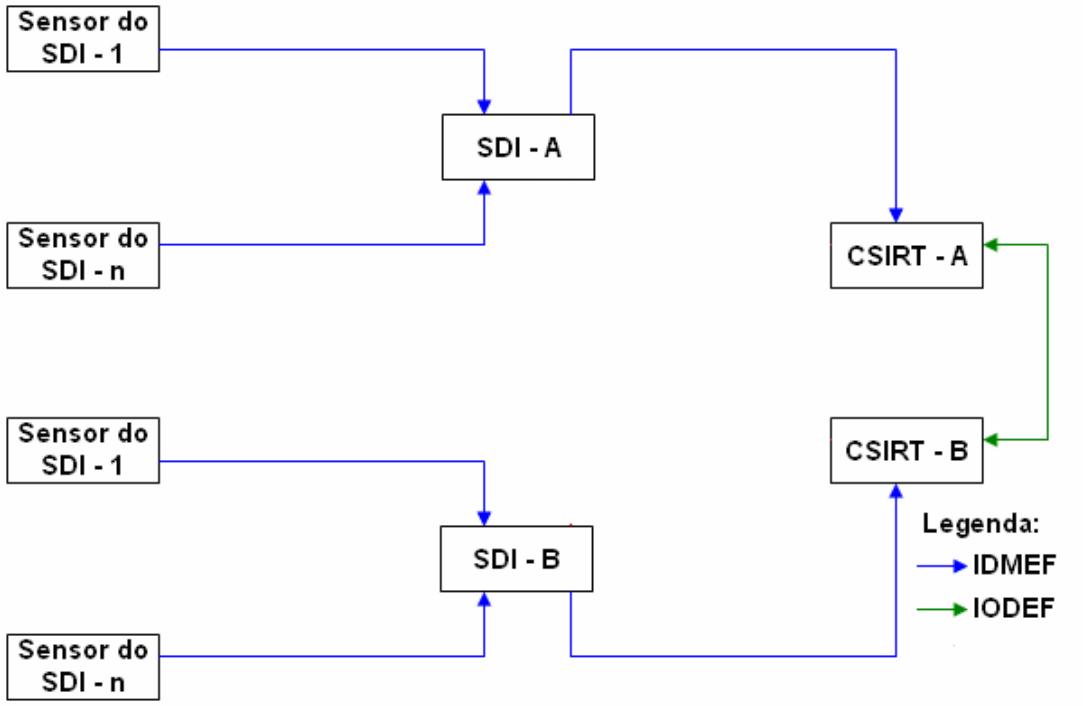

Figura 2.3. Utilização dos Formatos de Mensagens para Compartilhamento de Informações de Segurança.

O CAP é um formato genérico, não um padrão exclusivo para alertas referentes à segurança dos computadores e assim propicia um formato que permite a automação na geração dos mais diversos tipos de mensagens de alertas de emergência.

A Figura 2.3 mostra onde os diversos formatos para compartilhamento de informações de segurança são utilizados. O formato CAP não é ilustrado na Figura 2.3, pois pode conter mensagens em quaisquer níveis.

2.3 Common Vulnerabilities and Exposures (CVE)

O CVE (*Common Vulnerabilities and Exposures*) aspira descrever e nomear todos os fatos publicamente conhecidos sobre os sistemas computadorizados, que poderiam permitir que alguém violasse uma política razoável da segurança para esse sistema. Freqüentemente, estes fatos são denominados vulnerabilidades [19].

Considerando que o termo "vulnerabilidade" tem diversos usos, necessitou-se estabelecer uma maneira de fazer a distinção deste quando é apropriado. Por esta razão, foi introduzido o termo "exposição" para permitir a referência aos fatos relacionados à segurança que podem não ser considerados por todos como uma vulnerabilidade.

Em uma tentativa independente de distinguir os múltiplos aspectos aos quais o termo "vulnerabilidade" pode ser aplicado, o CVE identifica "vulnerabilidades universais", isto é, aqueles problemas que são considerados normalmente como vulnerabilidades dentro do contexto de qualquer política razoável de segurança, e "exposições", isto é, os problemas que são considerados violações somente por algumas políticas de segurança.

Assim o CVE é uma lista ou dicionário de nomes padronizados para vulnerabilidades e outras exposições da segurança da informação, e objetiva padronizar os nomes de todas as vulnerabilidades e exposições conhecidas da área de segurança digital [19]. Utilizando nomes comuns é possível compartilhar dados entre bases de dados e ferramentas diferentes que até então não são facilmente integráveis.

2.4 Network Intrusion Detection System Based on Intelligent Agent (NIDIA)

O *Network Intrusion Detection System Based on Intelligent Agent* (Sistema de Detecção de Intrusões Baseado em Agentes Inteligentes) NIDIA [38,39,55] constitui-se na proposta de um SDI Multiagentes [63], capaz de gerar um índice de suspeita de ataque a partir da análise de dados coletados de *logs* de máquinas e de pacotes da rede. A escolha da arquitetura multiagentes para um SDI visa obter várias vantagens conforme listadas na seção 2.5.

A concepção do NIDIA é inspirada no modelo lógico do CIDF [14,58], possuindo para este fim, agentes com funções de: gerar eventos, analisar dados, realizar ações de resposta, manter a integridade do sistema, coordenar as atividades do SDI como um todo e atualizar as bases de conhecimento.

2.4.1 Arquitetura NIDIA

Baseado em agentes, o NIDIA é capaz de detectar e tratar incidentes de segurança a partir de dados coletados de *logs* de *hosts* e pacotes de tráfego de rede, gerar um índice de suspeita de ataque e então executar

ações de contramedidas. Por ser multiagentes, pode-se ter um domínio administrativo composto por várias redes locais, monitorados por apenas um SDI NIDIA e diversos Agentes Sensores instalados em pontos estratégicos destas redes, conforme mostrado na Figura 2.4.

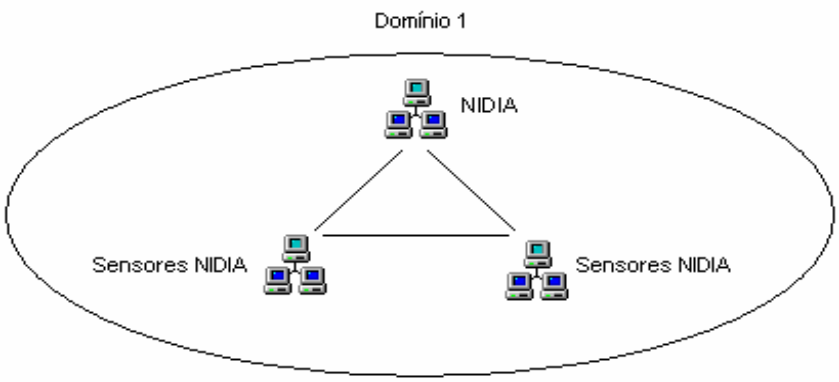

Figura 2.4. Estratégia de Monitoramento do NIDIA.

Para alcançar os objetivos a que se propõe, a arquitetura NIDIA é organizada em camadas. Cada uma das camadas é composta por uma sociedade de agentes que juntas cooperam, e formam o NIDIA. O Módulo Gerenciador de Informações [15,16] proposto neste trabalho foi desenvolvido para realizar as tarefas pertinentes às camadas de atualização e armazenamento, porém, por ser implementado como uma sociedade de agentes independentes pode ser utilizado por qualquer SDI baseado em agentes, como será descrito no Capítulo 4. Apresenta-se, na Figura 2.5, o modelo em camadas do SDI NIDIA.

O detalhamento das responsabilidades de cada camada que compõe a arquitetura do SDI NIDIA é descrita a seguir:

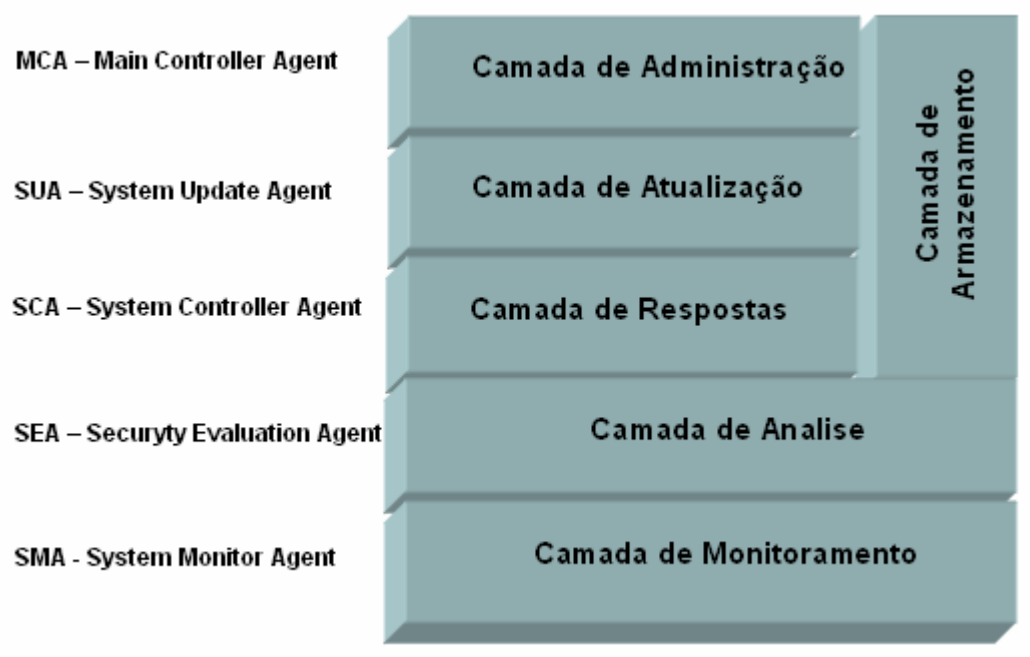

Figura 2.5. Arquitetura em Camadas do NIDIA.

- **Camada de Monitoramento**: camada responsável por capturar os eventos ocorridos e repassar informações sobre o mesmo para a camada de análise. Nesta camada estão localizados os agentes SMA (*System Monitoring Agent*). Agentes sensores que realizam o monitoramento, sendo de dois tipos:
    - Agentes sensores de rede - responsável por capturar os pacotes que estão trafegando na rede;
    - Agentes sensores de *host* - trabalham coletando informações em tempo real de um *host* em particular e disponibilizando-as para análise;
- **Camada de Análise**: nesta camada é realizada a análise dos eventos recebidos da camada de monitoramento. Executa a formatação dos eventos coletados de forma que possam ser identificados padrões de ataques e posteriormente a confirmação de

um verdadeiro ataque. Os agentes SEA (*Security Evaluation Agent*) compõem a camada de análise;

- **Camada de resposta**: tem como função disparar contramedidas caso um problema de segurança seja detectado. Para a tomada de decisão devem ser utilizadas informações previamente tratadas para determinar a ação e sua estratégia de execução. Nesta camada estão localizados os agentes SCA (*System Controller Agent*);
- **Camada de atualização**: camada responsável pela atualização das bases de informações. Ela terá também a responsabilidade de manter a integridade e consistência das informações armazenadas. Agentes do tipo SUA (*System Updating Agent*) executam as tarefas inerentes a esta camada;
- **Camada de administração**: a administração e integridade de todos agentes do sistema ficam a cargo desta camada. Aqui estão localizados os agentes MCA (*Main Controller Agent*);
- **Camada de armazenamento**: A responsabilidade de manter de forma persistente as informações provenientes das demais camadas é a atribuição desta camada. Nesta não existem agentes, ela foi idealizada para comportar as bases de dados utilizadas pelo NIDIA.

2.5 Sistemas Multiagentes

Em [44], um agente de software é definido como uma entidade de software autônoma que pode interagir com o ambiente. Esta autonomia significa que ele pode interagir de forma ativa com outras entidades, incluindo humanas, máquinas e outros agentes de software em vários ambientes e sobre

várias plataformas. Neste trabalho usar-se o termo “agente” para referir-se a “agente de software”.

A simples interação entre agentes não é suficiente para construir uma sociedade de agentes. Para tanto, necessita-se de agentes que podem ser coordenados para cooperação, competição, ou uma combinação de ambos. Esta "sociedade" de agentes é chamada de Sistemas Multiagentes ou MAS (*Multi-Agents Systems*). Sistemas Multiagentes, então, são sistemas compostos de agentes coordenáveis e dos relacionamentos que existem entre eles [44].

Aplicado a SDIs, vários projetos[35,57,59] têm sido desenvolvidos usando tecnologia de Sistemas Multiagentes. Dentre estes, além do NIDIA, pode-se citar o AAFID[57], o JAM[59] e o SPARTA[35]. Como vantagens deste paradigma para construção de SDIs, pode-se listar:

- 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, utilizando-se agentes especializados para um computador ou para uma rede em particular;
- Maior tolerância a falhas que sistemas monolíticos que possuem um único ponto crítico de falhas;

- A capacidade de criar agentes com relativa facilidade, e a de se utilizar artifícios como migração, fraca ou forte, pode incorporar ao ambiente maior resistência à falhas;
- Alto potencial de escalabilidade pela adição de novos agentes para manter o desempenho exigido em sistemas em expansão;
- Características como mobilidade pode ser usada para tornar o ambiente mais flexível e dinâmico;
- Pode-se ter um domínio administrativo composto por várias redes locais, monitorados por apenas um IDS e diversos agentes sensores instalados em pontos estratégicos da rede e em *hosts* específicos monitorando e enviando informações ao centro de controle do SDI.

2.6 Web Services

Fornecem padrões para comunicação entre diferentes aplicações que executam em diferentes plataformas.

Web Services [67] são aplicações modulares que podem ser publicadas, localizadas e invocadas através da Internet. Desta forma para um cliente acessar um serviço, este apenas precisa conhecer a definição do serviço, não sendo necessário conhecer como o serviço foi implementado. De outra forma, clientes e servidores não precisam ser escritos na mesma linguagem de programação.

Existem três componentes principais no ambiente de *Web Services* [67], todos eles são baseados no XML (*eXtensible Markup Language*): SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*) e UDDI (*Universal Description, Discovery, and Integration*).

Pode-se destacar dois usos principais para *Web Services* [68]:

- Reutilização de componentes de aplicações: Aplicações diferentes necessitam de componentes comuns freqüentemente, então porque implementar um componente que já está implementado? Os *Web Services* podem oferecer componentes já existentes como serviços. Idealmente, haverá somente um tipo de cada componente, e qualquer aplicação poderá usá-los;
- Conexão entre aplicações: Os *Web Services* auxiliam na resolução do problema de interoperabilidade, fornecendo às diferentes aplicações um meio para trocar informações.

2.7 Extensible Markup Language (XML)

É um formato de texto simples, muito flexível derivado de SGML (ISO 8879). Projetado originalmente visando os desafios de publicações eletrônicas em larga-escala. XML[66] está ocupando um papel cada vez mais importante na troca de grande variedade de dados em aplicações WEB.

Desenvolvido para descrever dados e focar no significado dos mesmos, XML pode ser usado para executar varias operações, dentre as quais podem ser destacadas [65]:

- Transformação: no mundo real, os sistemas computadorizados manipulam bases de dados que contêm dados em formatos incompatíveis. Um dos maiores desafios para os desenvolvedores é a troca de dados entre tais sistemas. O uso de XML pode reduzir extremamente esta complexidade criando dados em formatos que podem ser lidos por tipos diferentes de aplicações;

- Compartilhamento: com XML os dados são armazenados em formato de texto simples, assim, uma independência de software e hardware é fornecida para o compartilhamento de dados. Assim torna-se muito mais fácil criar dados que aplicações diferentes podem manipular. Outro aspecto importante é a fácil atualização dos sistemas para usar novos sistemas operacionais, servidores e navegadores de internet;
- Armazenamento: XML pode também ser usado para armazenar dados em arquivos ou bancos de dados. As aplicações podem ser escritas para armazenar e recuperar informações que foram armazenadas. E aplicações genéricas podem ser usadas como *front-end*.

2.8 Conclusões

Este Capítulo se deteve em uma introdução sobre as tecnologias usadas na solução do problema apresentado neste trabalho. Mostrou-se ainda, uma visão geral dos Sistemas de Detecção de Intrusão, enfatizando-se o SDI NIDIA descrito e detalhado através da apresentação de suas camadas.

Importantes propostas de formatos para troca de mensagens de informações de segurança, IDMEF, IODEF e CAP, também foram apresentadas e comparadas neste Capítulo.

3 PADRONIZAÇÃO DAS BASES DE DADOS

Antes de propor um serviço que gerencie dados, é necessário dizer quais dados necessitam ser gerenciados. Nesta pesquisa classifica-se os conjuntos de informações ou bases de dados usados por um SDI (Sistema de Detecção de Intrusão) considerando primeiramente a fonte geradora, ou seja, o próprio SDI e/ou uma entidade externa, e posteriormente agrupa-se os dados de acordo com a finalidade.

É importante salientar que em trabalhos anteriores [21,39,47] já haviam sido sugeridos alguns repositórios de dados que o NIDIA (*Network Intrusion Detection System Based on Intelligent Agent*) necessitaria para armazenar os dados necessários à execução de suas ações. Porém, nenhum padrão de formato, ou quais informações deveriam ser armazenadas em cada base de dados, foram determinados. Como o objetivo é desenvolver uma infra-estrutura que proporcione o compartilhamento e o gerenciamento das informações geradas por um SDI, este trabalho é iniciado com a padronização os dados a serem compartilhados e gerenciados.

Assim, neste Capítulo apresenta-se uma proposta de padronização das bases de dados utilizadas por um SDI. Inicialmente listam-se alguns requisitos desejáveis que a padronização proposta visa alcançar; em seguida mostra-se como as bases de dados foram organizadas, e por fim, descrevem-se os repositórios de dados.

Neste Capítulo, apresenta-se ainda um modelo proposto para troca de mensagens entre SDIs, concebido a partir do modelo CAP (*Common Alerting Protocol*)[11].

3.1 Requisitos para Padronização das Bases de Dados

Este grupo de requisitos, listados a partir da observação das características dos SDIs e da necessidade dos mesmos em exportar e importar informações, referem-se a qualidades que devem ser providas com a padronização das bases de dados. São estes os requisitos:

- Prover independência dos dados quanto ao tipo de SDI: SDIs que executem detecção por abuso e/ou anomalia, monitoramento a nível de rede e/ou *host*, análise de informações em *bath* e/ou em tempo real, que possuam reações passivas e/ou ativas devem ser igualmente contemplados;
- Garantir estensibilidade: o formato de dados usado para armazenamento deve estar pronto ao surgimento de novas tecnologias de detecção de intrusão, estando apto a incluir novas informações sem maiores impactos;
- Disponibilizar suporte a internacionalização: provendo assim independência de detalhes geográficos e culturais;
- Prover nível de detalhamento adequado para armazenamento de informações forense: informações geradas a partir de *logs* do SDI, que possuem nível de detalhamento variado dependendo do SDI;
- Garantir flexibilidade: permitindo fácil adequação;
- Prover capacidade de representar todas as informações pertinentes a um SDI:
    - Informações sobre incidentes de intrusões;
    - Informações forenses;

- Informações de assinaturas de ataque;
- Informações de ações de resposta;
- Informações de como as ações podem ser executadas dentro da política de segurança da instituição.

3.2 Informações Gerenciáveis

A partir de estudos realizados em vários projetos de SDI [36,38,49,57,59], constatou-se que são quatro os grupos de informações necessárias a um SDI: informações sobre assinaturas de ataque, informações de ações de resposta, informações forenses e informações de estratégias para execução de ações de resposta.

As informações sobre assinaturas de ataque e sobre ações de resposta são predominantemente fornecidas por entidades externas ao SDI, como por exemplo, os Grupos de Respostas a Incidentes de Segurança em Computadores ou CSIRT (*Computer Security Incident Response Team*). Já as informações forenses e informações de estratégias são normalmente geradas internamente ao SDI.

Para a padronização das bases de dados utiliza-se XML, ou seja, o formato que os dados são armazenados em meio permanente é XML. Visando com isso garantir maior capacidade de compartilhamento e interoperabilidade entre SDIs.

3.3 Informações de Assinaturas de Ataques

A detecção de intrusão é definida por Mukherjee [41] como o problema de identificar indivíduos que estão usando um sistema de

computador sem autorização, e outros que têm acesso legítimo ao sistema, mas estão abusando de seus privilégios.

O processo de detecção de atividades agressoras ocupa um papel fundamental nas funções de um SDI [52]. A detecção é a primeira etapa da resposta [40]. Portanto, para que o sistema possa tomar as medidas necessárias para proteger a rede e começar o processo de rastrear o possível invasor, é necessário que o ataque seja identificado de forma rápida e precisa.

Essa base armazena padrões de ataques e é utilizada para a detecção de intrusão. Possui informações que são obtidas a partir de fontes especializadas na internet, e deve necessariamente estar atualizada a fim de prover ao SDI informações consistentes para a detecção de ataques.

3.3.1 Estrutura e dicionário de dados da base de intrusões

A Figura 3.1 mostra a estrutura da base de dados de intrusões que armazena as assinaturas de ataques. Este repositório foi concebido para auxiliar a detecção de ataques em SDIs que executam detecção por abuso comparando informações coletadas em rede ou *host* com assinaturas de ataques conhecidas e previamente armazenadas neste repositório.

A seguir descrevem-se os elementos que compõem o repositório de dados de intrusão. Como o formato adotado é XML, procede-se a descrição por elementos e sub-elementos nas subseções que se seguem.

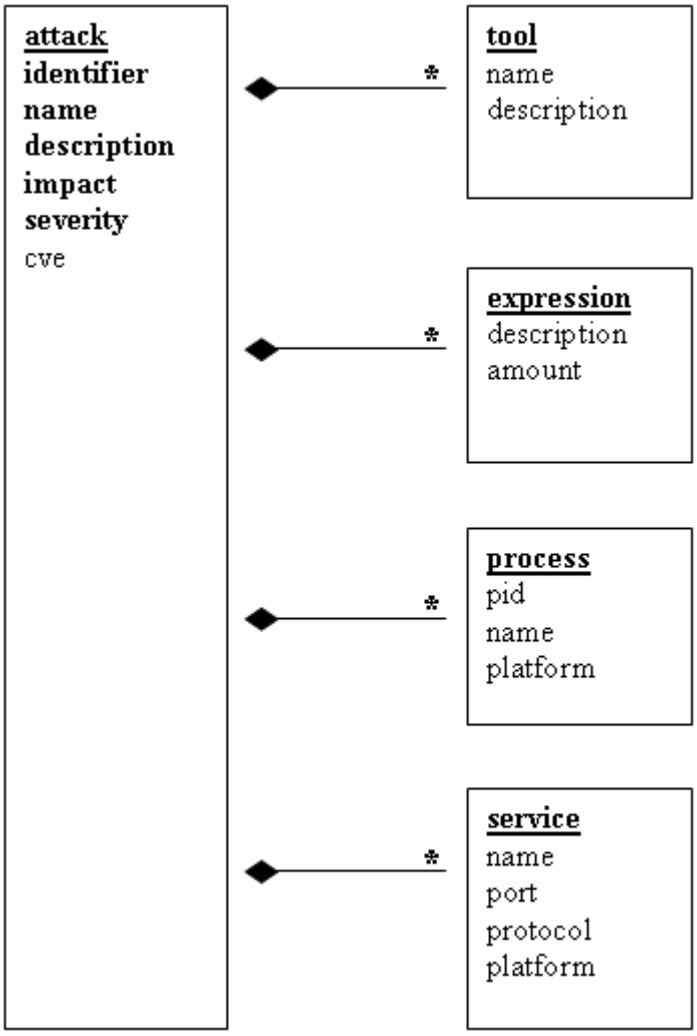

Figura 3.1. Formato da Base de Assinaturas de Ataques.

3.3.1.1 Elemento <attack> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
attack	Container para todas as partes da informação de intrusão (obrigatório)	
identifier	O identificador de um registro na base de intrusões (obrigatório)	(1) Identifica uma intrusão de forma única.
name	Nome da intrusão (opcional)	(1) Pode coincidir com o nome dado pelo CVE ou não
description	Texto descrevendo a intrusão (opcional)	

impact	Texto descrevendo os impactos da intrusão no sistema (opcional)	
severity	Código especificando a severidade da intrusão (obrigatório)	(1) Valores dos códigos: "*extreme*" – ameaça extraordinária "*severe*" – ameaça significativa "*moderate*" – possível ameaça "*minor*" – ameaça mínima "*unknown*" – ameaça desconhecida
cve	Nome da intrusão conforme padrão CVE (opcional)	

3.3.1.2 Elemento <tool> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
tool	Container para todas as partes componentes do sub-elemento <tool> (opcional)	(1) Múltiplas ocorrências são permitidas em único <attack>
name	Nome da ferramenta (obrigatório)	
description	Texto descrevendo a ferramenta utilizada para executar a intrusão (opcional)	

3.3.1.3 Elemento <expression> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
expression	Container para todas as partes componentes do sub-elemento <expression> (obrigatório)	(1) Múltiplas ocorrências são permitidas em um único <attack>
description	Texto contendo a assinatura de ataque (obrigatório)	

amount	Número de ocorrências do componente <description> que evidencia um ataque (obrigatório)	

3.3.1.4 Elemento <process> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
process	Container para todas as partes componentes do sub-elemento <process> (opcional)	(1) múltiplas ocorrências são permitidas em um único <attack>
pid	Código que identifica o processo. (opcional)	(1) Processo usado na execução da ação de intrusão
name	Nome do processo. (obrigatório)	(1) Texto que descreve o nome do processo (2) Caso haja a necessidade de usar espaços em branco, este campo deve está entre aspas duplas.
platform	Código específicando o nome da plataforma do processo de sistema utilizado no ataque. (opcional)	(1) Valores dos Códigos: "s*parc*" – Plataforma *Sparc Sun.* "x86" – Plataforma *Intel.* "w*indows*" – *Microsoft Windows.* "s*olaris*" – Plataforma *Solaris* da *Sun.* "linux" – Plataforma Linux.

3.3.1.5 Elemento <service> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
service	Container para todas as partes componentes do sub-elemento <service> (opcional)	(1) Múltiplas ocorrências são permitidas em um único <attack>
name	Nome do serviço utilizado para executar a intrusão (obrigatório)	(1) Texto que descreve o nome do serviço (2) Caso haja a necessidade de usar espaços em branco, este campo deve está entre aspas duplas.

port	Número inteiro entre 0 e 65535 que indica o número da porta que o serviço utiliza. (opcional)	
protocol	Código que específica o nome do protocolo associado à porta que o serviço utiliza. (opcional)	(1) Valores dos códigos: “tpc” – protocolo TCP “udp” – protocolo UDP
platform	Código específicando o nome da plataforma do serviço de sistema utilizado no ataque. (opcional)	(1) Valores dos Códigos: “s*parc*” – Plataforma *Sparc Sun.* “x86” – Plataforma *Intel.* “w*indows*” – *Microsoft Windows.* “s*olaris*” – Plataforma *Solaris* da *Sun.* “linux” – Plataforma Linux.

3.4 Informações de Ações de Resposta

Uma resposta pode ser definida como um conjunto de ações que um SDI pode executar quando ele detecta uma intrusão [6]. A depender da ação e da política de segurança adotada, estas ações podem ser desencadeadas de forma automática ou mediante autorização e monitoramento do administrador. Dentre as ações que podem ser executadas, podem-se destacar [50]:

- Coleta de informações adicionais: todas as ações do invasor são rastreadas com a finalidade de:
    - Gerar relatórios fornecendo uma visão geral do que aconteceu, possibilitando a tomada de outras ações de forma eficiente;
    - Alimentar a base de dados forense;
- Geração de alertas: o sistema envia avisos de invasão ou tentativa de intrusão ao administrador e/ou a uma entidade lógica responsável por analisar a situação e tomar as futuras ações necessárias;

- Criar cópias de segurança: ataques contra a integridade do sistema podem ter suas conseqüências reduzidas se forem criadas cópias de segurança. Além disso, cópias atualizadas permitem a restauração do sistema e a comparação de arquivos;
- Alterar as configurações do ambiente: ao detectar um ataque o SDI pode alterar as configurações do ambiente a ser protegido com a finalidade de evitar maiores danos. Essas alterações compreendem ações como: quebra da conexão do invasor; remoção de permissões do usuário; alteração de permissões do usuário; reconfiguração de tabelas de roteamento; reconfiguração de regras do *firewall*; derrubar a máquina alvo do ataque;
- Executar ações contra o invasor: esse nível de resposta a uma intrusão ou tentativa de intrusão é sem duvida a ação mais forte que se pode executar quando um alerta é disparado. No entanto, Essa ação deve ser executada com cautela, pois na maioria das vezes o IP de origem detectado como sendo usado pelo atacante pode ser falso, levando o sistema de defesa a atacar um usuário inocente.

As informações armazenadas nessa base de dados devem ser utilizadas em conjunto com as informações referentes às estratégias de execução dessas ações.

3.4.1 Estrutura e dicionário de dados da base de ações de resposta

Este repositório foi projetado para auxiliar a execução das ações de respostas que são desencadeadas pelo SDI após a detecção de uma intrusão.

A Figura 3.2 mostra a estrutura da base de dados de ações de respostas.

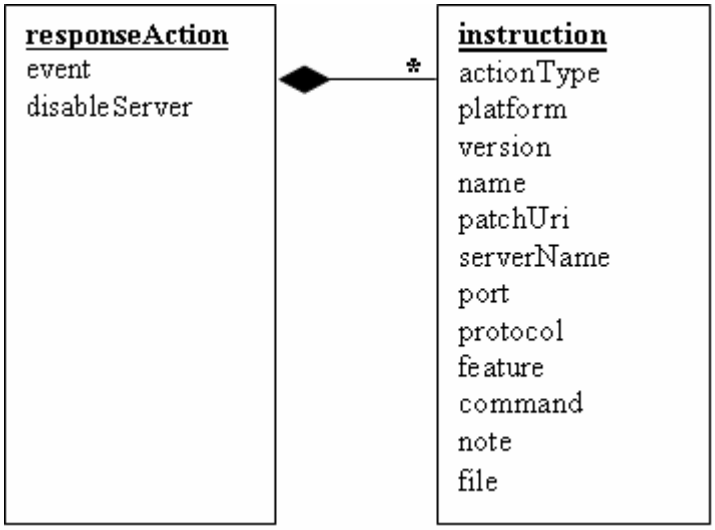

Figura 3.2. Formato da Base de Ações de Resposta.

A seguir descrevem-se os elementos que compõem o repositório de ações de resposta. Como já mencionado o formato adotado para o armazenamento é XML, procede-se então, à descrição por elementos e sub-elementos nas subseções que se seguem.

3.4.1.1 Elemento <responseAction> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
responseAction	Container para todas as partes da informação de ações de resposta (obrigatório)	
event	O identificador de um evento de intrusão ao qual a ação de resposta deve ser disparada (obrigatório)	(1) Essa informação deve coincidir com o sub-elemento <identifier> do elemento <attack> da base de dados de intrusão
disableServer	Grupo que lista os nomes dos servidores que devem ser desativados até que o evento identificado seja neutralizado (opcional)	(1) Caso haja múltiplas instâncias elas devem ser separadas por espaço em branco. Nomes de servidores que possuem espaço em branco devem está entre aspas duplas.

3.4.1.2 Elemento <instruction> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
instruction	Container para todas as partes componentes do sub-elemento <instruction>. (obrigatório)	(1) Múltiplas ocorrências são permitidas em um único <responseAction>.
actionType	Define a ação a ser executada (obrigatório)	(1) Os valores de *actionType* são: *"applyPath", "removePath", "blockAcces", "disableFeature", "modifyFile"* e *"permissionChange"* .
platform	Define a plataforma do sistema ao qual a ação pode ser executada (opcional)	(1) Para as ações *applyPath* e *removePath* este elemento deve ser preenchido
version	Identifica a versão do sistema no qual a ação pode ser executada (opcional)	(1) Elemento que pode ser preenchido caso a ação a ser executada seja *applyPath* ou *removePath*.
namePlatform	Texto usado para complementar a identificação da plataforma (opcional)	(1) Uma string que identifica unicamente a versão do sistema no qual a ação pode ser executada. (2) Elemento usado para complementar as informações caso a ação a ser executada seja *applyPath* ou *removePath.*
patchUri	Identificador do *hiperlink* para o arquivo de correção (obrigatório)	(1) Elemento utilizado caso a ação a ser executada seja *applyPath.*
pathaName	Identificador da correção do sistema que deve ser desinstalado (obrigatório)	(1) Uma string que identifica unicamente o nome da correção do sistema. (2) Elemento utilizado caso a ação a ser executada seja *removePath.*
serverName	Identificador do servidor onde portas ou protocolos deverão ser bloqueados (obrigatório)	(1) Uma string que identifica unicamente o nome de um servidor que deve ser alvo da ação. (2) Elemento utilizado caso a ação a ser executada seja *blockAcces, disableFeature,*

		modifyFile ou permissionChange.
port	Número inteiro entre 0 e 65535 que indica o número da porta que deve ser bloqueada (obrigatório)	(1) Elemento utilizado caso a ação a ser executada seja *blockAcces*. (2) Número de portas de acordo com a específicação da IANA (Internet Assigned Numbers Authority).
protocol	Código que específica o nome do protocolo associado à porta que deve ser bloqueada. (obrigatório)	(1) Valores dos códigos: “tpc” – protocolo TCP “udp” – protocolo UDP (2) Elemento utilizado caso a ação a ser executada seja *blockAcces*.
feature	Identificador do nome do serviço que será desabilitado. (obrigatório)	(1) Uma string que identifica unicamente o nome do serviço ou parte dele. (2) Elemento utilizado caso a ação a ser executada seja *disableFeature*.
command	Texto contendo os comandos necessários para desabilitar um serviço, alterar um arquivo ou alterar as permissões de acesso aos recursos do sistema. (obrigatório)	(1) elemento que deve ser preenchido para as ações *disableFeature, modifyFile, permissionChange*.
note	Texto explicativo do serviço a ser desabilitado e possíveis efeitos no sistema, das alterações em arquivos e suas prováveis conseqüências ou das alterações de acesso e seus efeitos. (opcional)	(1) Elemento que poderá ser preenchido para as ações *disableFeature, modifyFile, permissionChange*.
file	Nome do arquivo a ser modificado contendo o seu caminho completo. (obrigatório)	(1) Elemento que deve ser preenchido para a ação *modifyFile*

3.5 Informações de Estratégias de Execução de Ações de Resposta

Esse conjunto de dados armazena informações de como as ações devem ser executadas dentro da política de segurança da instituição. Devem ser usados para disciplinar a reação do SDI levando em consideração uma série de fatores que são peculiares a cada organização.

Estatísticas sobre ataques, reações do SDI, estratégias adotadas e resultados obtidos devem estar disponíveis para que o componente do SDI responsável pela adoção da estratégia siga o “melhor caminho”.

Essa base de dados deve ser alimentada, preferencialmente, pelo administrador responsável pela segurança, e é de vital importância para que as ações de respostas a ataques sejam tomadas de maneira eficiente.

3.5.1 Estrutura e dicionário de dados da base de estratégias

A Figura 3.3 mostra a estrutura do repositório de informações de estratégias. Em seguida descrevem-se os elementos e sub-elementos que compõem a base de dados de estratégias de ações de resposta detalhando-se o sentido de cada campo que a compõem.

Figura 3.3. Formato da Base de Estratégias de Ações de Resposta.

3.5.1.1 Elemento <strategy> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
strategy	Container para todas as partes da informação de estratégia de ações de resposta. (obrigatório)	
actionType	Define a ação a ser executada. (obrigatório)	(1) Essa informação deve coincidir com o componente <action> do sub-elemento <restriction> do elemento <responseAction> da base de dados de ação de respostas.
event	O identificador de um evento de intrusão ao qual a ação de resposta deve ser disparada (obrigatório)	(1) Essa informação deve coincidir com o componente <event> do elemento <responseAction> da base de dados de ação de respostas.
advice	Código especificando a estratégia a ser adotada para a execução da ação escolhida.	(1) Valores dos códigos: "*immediate*" – executa imediatamente a ação escalonada. "*disconnectUser*" – caso a ação seja específicamente sobre um servidor da rede, desconecta todos os usuários sem salvar as transações pendentes e executa a ação. "*requestDisconnect*" – solicita a desconexão e o encerramento das atividades dos usuários correntes e logo após executa a ação escolhida. "*schedule*" – agenda a ação de resposta para um horário pré-determinado. "*requestAutorization*" – contacta o administrador e solicita autorização para executar a ação pré-definida.

3.6 Informações Forenses

O registro de danos causados por ataques bem-sucedidos e/ou tentativas de ataques, são informações que podem ser úteis na identificação de tentativas de ataques provenientes de uma mesma origem ou domínio, ou ainda serem utilizadas em investigações futuras.

Dentre as informações que devem ser armazenadas têm-se: *hora, data, domínio, usuário, evento detectado, ação tomada, resultado obtido, ip de origem do evento, ip de destino do evento etc.* Esse grupo de dados é de fundamental importância para investigações e possíveis aplicações de penas legais ao usuário autor do ataque.

Outra importante aplicação desse grupo de dados é a geração de estatísticas que podem ser usadas como fonte para o refinamento e otimização do SDI. Estas estatísticas devem ser levadas em consideração no momento de definir a ação de contra-ataque e a estratégia adotada para execução da mesma.

3.6.1 Formato para troca de Informações Forenses entre SDIs

Como mencionado no Capítulo 2, o CAP é um formato, em XML, aberto e não proprietário de mensagens digitais para todos os tipos de alertas e notificações, sendo compatível com *Web Services* e com suporte a criptografia e assinatura digital.

Desta forma, propõe-se um formato de mensagem para troca de dados forenses como uma extensão do CAP. A Figura 3.4 traz uma visão macro do modelo proposto para troca de informações forenses a partir do modelo CAP.

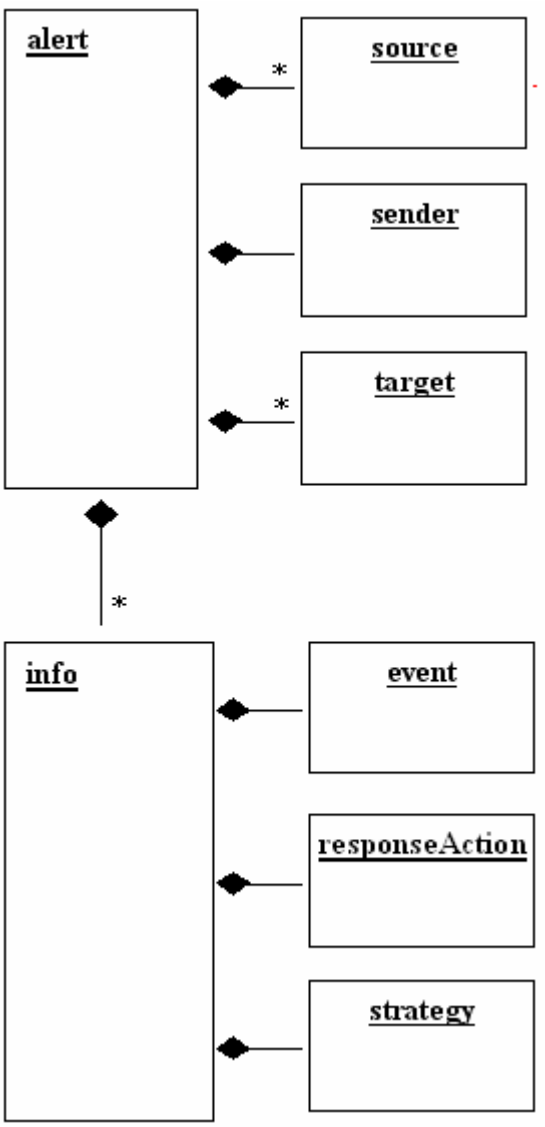

Figura 3.4. Visão Macro do Modelo Proposto para Troca e Armazenamento de Informações Forense.

Pode-se citar as seguintes razões para a escolha do CAP: a extensão do mesmo evita a utilização de um padrão exclusivo para alertas referentes à segurança dos computadores e propicia um formato que permite a representação e automação da troca de informações forenses. Além disso, tem-se também, o fato do NIDIA já dispor de um formato de mensagem estendida a partir do CAP para a obtenção de informações de ações de respostas [48]. A extensão aqui proposta foi realizada por meio da adição de elementos que possibilitem a identificação do evento detectado, das ações de

resposta executadas em contrapartida e das estratégias adotadas para execução das ações de resposta. Como o CAP é um modelo genérico, alguns campos foram retirados do modelo original por não ser necessário para o tipo específico de mensagem a ser suportada.

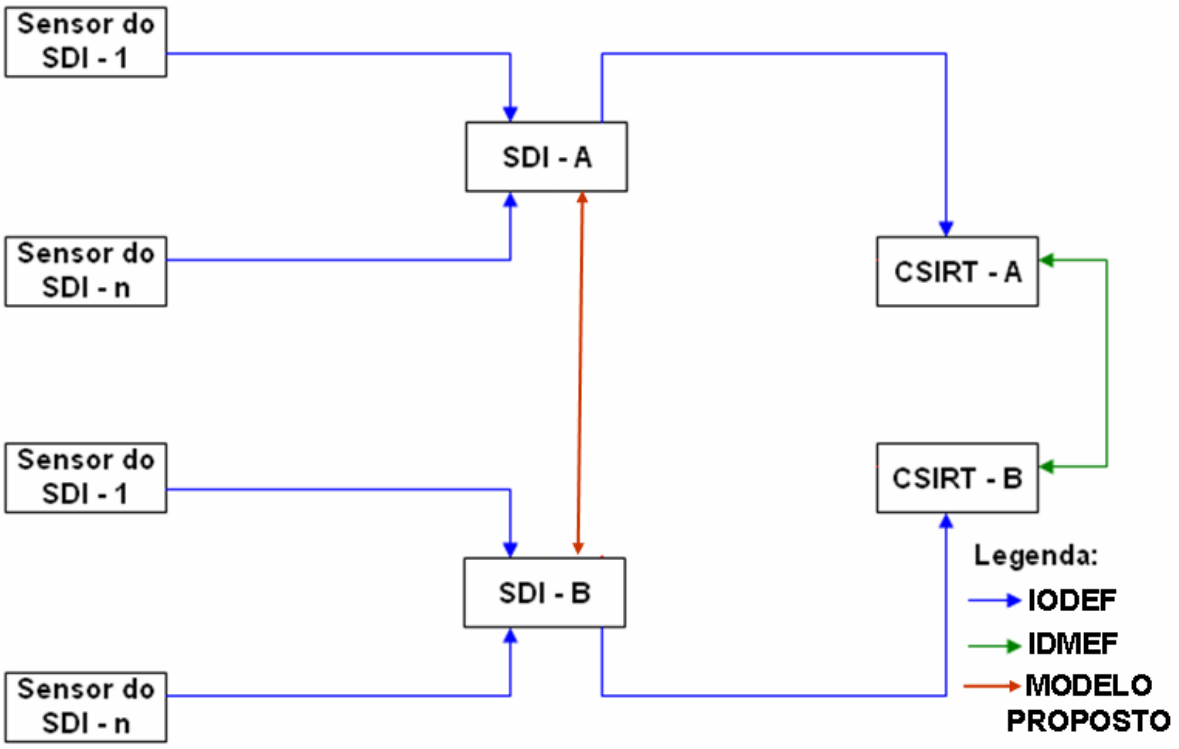

Figura 3.5. Comparativo do escopo do Modelo Proposto para Troca de Dados Forenses com os Modelos IODEF e IDMEF.

O detalhamento do modelo proposto para troca de mensagens, contendo informações forenses, será realizado juntamente com a apresentação do modelo proposto para armazenamento destas informações na subseção 3.6.2. Apresenta-se na Figura 3.5 um comparativo entre os modelos propostos para comunicação do IETF, IODEF e IDMEF, e o modelo proposto nesta dissertação para troca de dados forenses entre SDIs.

3.6.2 Estrutura e dicionário de dados da base forense

Para formatação da base de dados utilizou-se a mesma extensão CAP proposta para troca de informações.

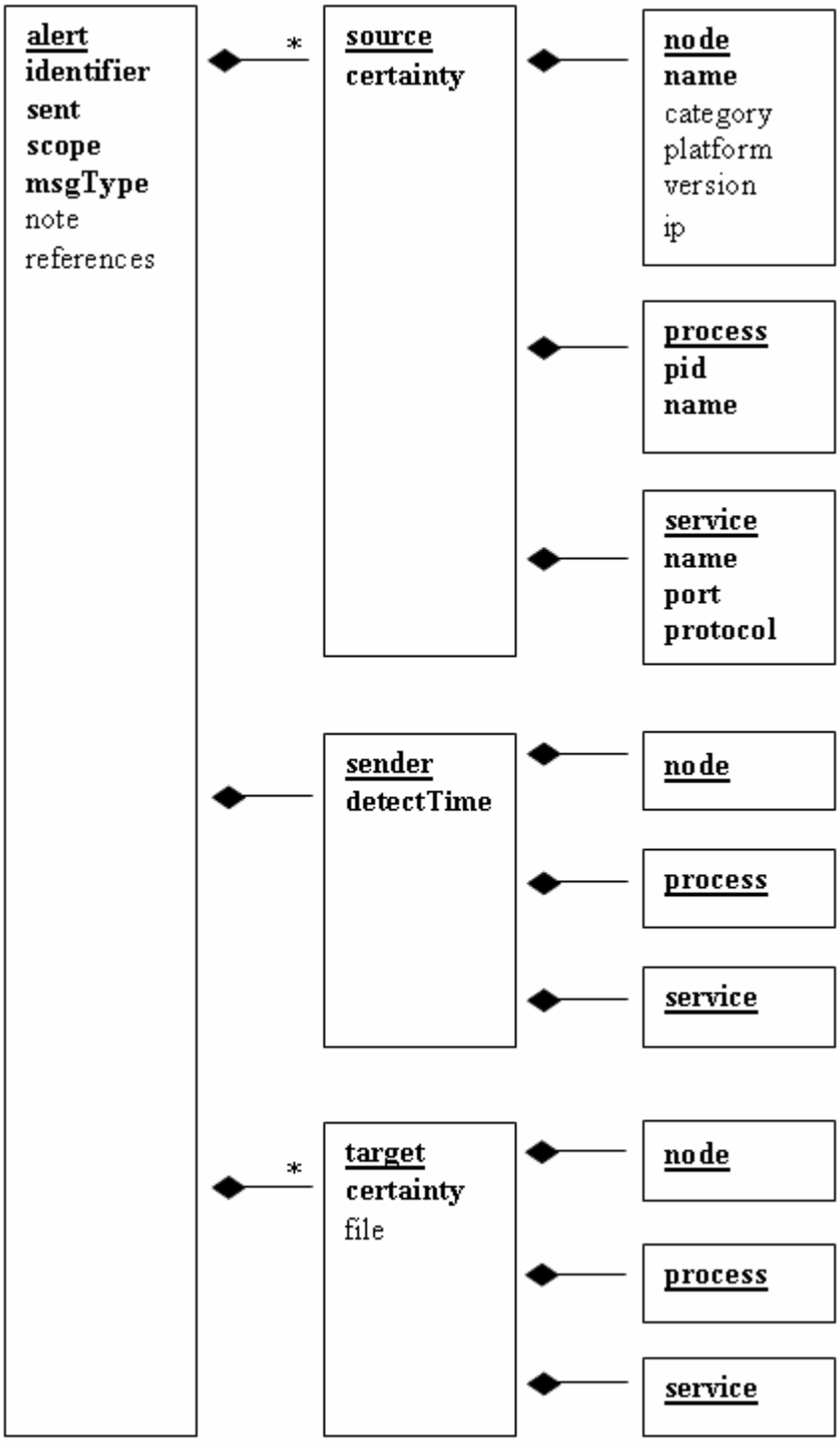

Figura 3.6.(a) Formato da Base de Dados Forenses.

Isto garante que o formato em que os dados são armazenados, seja o mesmo utilizado nas mensagens quando do compartilhamento de informações entre SDIs.

A estrutura desta base de dados e mensagens, contendo dados forenses, baseada na estrutura da mensagem de alerta CAP consiste de um segmento “*alert*” que conterá um ou mais segmentos “*source*”, um ou mais segmentos “*target*”, um segmento “*sender*” e um ou mais segmentos “*info*”. O segmento “*info*” por sua vez, conterá um segmento “*event*”, um segmento “*responseAction*” e um segmentos “*strategy*”. Essa estrutura é mostrada na Figura 3.4.

As Figuras 3.6.a e 3.6.b detalham a base de dados forenses. Os sub-elementos “*event*”, “*responseAction*” e “*strategy*” não serão detalhados, pois possuem os mesmos componentes, com a mesma semântica, das bases anteriormente descritas nas subseções 3.2, 3.3 e 3.4. Os demais elementos e sub-elementos serão detalhados a seguir.

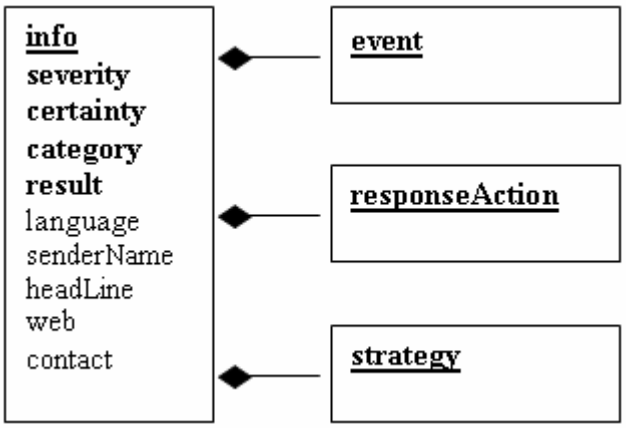

Figura 3.6.(b) Formato da Base de Dados Forenses.

3.6.2.1 Elemento <alert> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
alert	Container para todas as partes da informação de dados forenses. (obrigatório)	

identifier	O identificador de um dado forense. (obrigatório)	(1) Identifica um evento ocorrido de forma única
sent	A hora e data de geração da mensagem de alerta. (obrigatório)	(1) A data e hora são representadas no formato ISO 8601
scope	Código que define o tipo de distribuição pretendida para esta informação. (obrigatório)	(1) Valores dos Códigos: "Public"- Para disseminação geral. "Restricted" - Para disseminação apenas a usuários que possuam um requerimento operacional conhecido. "Private" - Para disseminação apenas a endereços específicados.
msgType	Código que denota a natureza de uma mensagem de alerta. (obrigatório)	(1) Valores dos Códigos: "*Alert*" - Informação inicial que requer atenção por parte dos destinatários. "*Update*" - Atualiza e substitui uma ou mais mensagens anteriores identificadas pelo elemento <*references*>. "*Cancel*" - Cancela uma ou mais mensagens anteriores identificadas pelo elemento <references>. "*Ack*" - Confirma o recebimento e a aceitação de uma ou mais mensagens identificadas pelo elemento <*references*>. "*Error*" - Indica a rejeição de uma ou mais mensagens identificadas pelo elemento <*references*>; uma explicação pode estar no elemento <*note*>.
note	Texto que descreve o propósito ou significado de uma mensagem de alerta. (opcional)	(1) Notas de mensagens são destinadas para uso com as mensagens de alerta do tipo "*Cancel*" e "*Error*".
references	Uma lista de mensagens anteriores referenciadas por uma mensagem de	(1) Identificador estendido de mensagem (na forma <*identifier*>/<*sender*>) de

	alerta. (opcional)	uma ou mais mensagens anteriores referenciadas por um alerta. (2) Se múltiplas mensagens são referenciadas, elas são separadas por espaços em branco.

3.6.2.2 Elemento <source> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
source	Container para todas as partes componentes do sub-elemento <source> da base de informação forense. (obrigatório)	(1) Múltiplas ocorrências são permitidas para um único elemento <alert>. (2) contém a identificação da fonte geradora do evento.
certainty	Código específicando o nível de confiabilidade da informação que define a fonte geradora do evento. (obrigatório)	(1) Valores dos Códigos: "*Very Likely*" - Altamente provável (p >~85%) ou certo. "*Likely*" - Provável (p >~50%). "*Possible*" - Possível mas não provável (p <=~50%). "*Unlikely*" – Improvável (p ~ 0). "*Unknown*" - Certeza desconhecida

3.6.2.3 Elemento <node> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
node	Container para todas as partes componentes do sub-elemento <node> da base de informação forense. (obrigatório)	(1) Contém a identificação do nó que originou o evento.
name	Identificação do nome do nó que originou o evento. (obrigatório)	(1) Uma *string* que identifica a máquina que originou o evento. (2) Caso haja a necessidade de usar espaços em branco, este campo deve está entre aspas duplas.

category	Identifica a categoria do nó. (opcional)	(1) Valores dos Código: “server” – o nó é um servidor. “pc” – o nó é um computador pessoal ou estação de trabalho.
platform	Código específicando o nome da plataforma do sistema que é utilizado no nó de origem. (opcional)	(1) Valores dos Códigos: “s*parc*” – Plataforma *Sparc Sun.* “x86” – Plataforma *Intel.* “w*indows*” – *Microsoft Windows.* “s*olaris*” – Plataforma *Solaris* da *Sun.* “linux” – Plataforma Linux.
version	Identificador da versão do sistema ao qual a mensagem de alerta se refere. (opcional)	(1) Um número ou “*string*” que identifica unicamente a versão de um sistema que é atribuído pelo fabricante. (2) Caso haja a necessidade de usar espaços em branco, este campo deve está entre aspas duplas.
ip	Identifica o protocolo *ip* do nó que originou o evento. (opcional)	

3.6.2.4 Elemento <process> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
process	Container para todas as partes componentes do sub-elemento <node> da base de informação forense. (obrigatório)	(1) Contém a identificação do nó que originou o evento.
pid	Código que Identificador o processo. (opcional)	(1) Processo usado na execução da ação de intrusão.
name	Nome do processo. (obrigatório)	(1) Texto que descreve o nome do processo (2) Caso haja a necessidade de usar espaços em branco, este campo deve está entre aspas duplas.

3.6.2.5 Elemento <service> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
service	Container para todas as partes componentes do sub-elemento <node> da base de informação forense. (obrigatório)	(1) Contém a identificação do nó que originou o evento.
name	Nome do serviço utilizado para executar a intrusão (obrigatório)	(1) Texto que descreve o nome do serviço. (2) Caso haja a necessidade de usar espaços em branco, este campo deve estar entre aspas duplas.
port	Numero inteiro entre 0 e 65535 que indica o numero da porta utilizada pelo serviço. (opcional)	
protocol	Código que específica o nome do protocolo associado à porta. (opcional)	(1) Valores dos códigos: "tpc" – protocolo TCP "udp" – protocolo UDP

3.6.2.6 Elemento <sender> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
sender	Container para todas as partes componentes do sub-elemento <sender> da base de informação forense. (obrigatório)	(1) Contém a identificação do emissor da mensagem de alerta dentro do sistema.
detectTime	Hora e data da identificação do evento de intrusão. (obrigatório)	(1) A data e hora são representadas no formato ISO 8601

Os sub-elementos *"node", "process"* e *"service"* dentro do sub-elemento "*sender*" não serão detalhados, pois possuem os mesmos componentes com a mesma semântica dos sub-elemento *"node", "process"* e

"service" pertencentes ao sub-elemento "*source*" detalhados anteriormente nas subseções 3.6.2.3, 3.6.2.4 e 3.6.2.5 respectivamente.

3.6.2.7 Elemento <target> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
target	Container para todas as partes componentes do sub-elemento <target> da base de informação forense. (obrigatório)	(1) Múltiplas ocorrências são permitidas para um único elemento <alert>. (2) Contém a identificação da fonte geradora do evento.
certainty	Código especificando o nível de confiabilidade da informação que define a o alvo da intrusão ou ataque. (obrigatório)	(1) Valores dos Códigos: "*Very Likely*" - Altamente provável (p >~85%) ou certo. "*Likely*" - Provável (p >~50%). "*Possible*" - Possível mas não provável (p <=~50%). "*Unlikely*" – Improvável (p ~ 0). "*Unknown*" - Certeza desconhecida
file	Identifica o arquivo alvo da intrusão ou ataque. (opcional)	(1) Componente usado no caso do ataque objetivar modificar ou excluir um arquivo. (2) Caso haja mais de um arquivo, as entradas devem estar separadas por um espaço em branco e valor deste componente entre aspas duplas.

Os sub-elementos *"node", "process"* e *"service"* dentro do sub-elemento *"target"* não serão detalhados, pois possuem os mesmos componentes com a mesma denotação dos sub-elemento *"node", "process"* e *"service"* pertencentes ao sub-elemento *"source"* já detalhados nas subseções 3.6.2.3, 3.6.2.4 e 3.6.2.5 respectivamente.

3.6.2.8 Elemento <info> e sub-elementos

Nome do Elemento	Definição	Notas ou Valor de Domínio
Info	Container para todas as partes componentes do sub-elemento <info> da base de dados forenses. (obrigatório)	(1) Múltiplas ocorrências são permitidas em um único <*alert*>. Se múltiplos elementos <*info*> do mesmo idioma se sobrepõem, a informação nos elementos posteriores podem ampliar mas possivelmente não anulam os valores correspondentes dos elementos anteriores.
severity	Código especificando a severidade do assunto de uma mensagem de alerta. (obrigatório)	(1) Valores dos Códigos: "*Extreme*" - Ameaça extraordinária a vida ou a propriedade. "*Severe*" - Ameaça significante a vida ou a propriedade. "*Moderate*" - Possível ameaça à vida ou à propriedade. "*Minor*" - Ameaça mínima a vida ou a propriedade. "*Unknown*" - Severidade desconhecida.
category	Código especificando a categoria do assunto de uma mensagem de alerta. (opcional)	(1) Como o CAP não prevê um código para problemas de segurança na Internet, criou-se um valor para essa categoria: "Internet" - Problemas de segurança na Internet (2) Múltiplas ocorrências são permitidas em um único elemento <*info*>.
result	Código que indica o resultado da(s) ação(s) executada(s) pelo sistema em decorrência do evento identificado. (obrigatório)	(1) Texto que descreve o resultado obtido. (2) Caso haja a necessidade de usar espaços em branco, este campo deve estar entre aspas duplas.
language	Código especificando o idioma do sub-elemento <*info*>. (opcional)	(1) Valores dos Códigos: identificadores de linguagem natural pela RFC 1766. (2) Se não estiver presente, o valor assumido é "en-US".

senderName	Texto que nomeia o criador de uma mensagem de alerta. (opcional)	Nome da agência ou autoridade que emitiu o alerta.
headline	Título da mensagem de troca de informações forenses. (opcional)	Um título breve.
Web	Identificador do *hyperlink* que apresenta informações adicionais relacionadas à mensagem de troca de informações forenses. (opcional)	Um URL completo para uma página HTML ou outro recurso de texto com informações adicionais ou de referência relacionados à mensagem de alerta.
contact	Texto descrevendo o contato para acompanhamento e confirmação de uma mensagem de troca de informações forenses. (opcional)	

3.7 Conclusões

Um SDI, como descrito, é um sistema pró-ativo que toma decisões baseado-se em informações. Deste modo é necessário que essas informações encontrem-se devidamente atualizadas, e é fortemente desejável que essas atualizações sejam realizadas de forma automática.

A interoperabilidade só é possível com a padronização. Assim, neste Capítulo apresentou-se uma proposta de organização dos dados em bases distintas e padronização dessas bases de dados, usando XML, para garantir que requisitos como *Independência quanto ao tipo de SDI*, *Estensibilidade*, *Flexibilidade* e *Capacidade de Representar todas as Informações Pertinentes a um SDI* sejam inseridos no ambiente.

Propôs-se também, neste Capítulo, um modelo para troca de informações forenses baseado no modelo concebido para troca de alertas de emergência CAP.

4 MODELO PROPOSTO

Um SDI (Sistema de Detecção de Intrusão) é uma ferramenta de segurança que deve tomar decisões considerando um conjunto de fatores que podem influenciar a ação a ser desencadeada em contra-ataque. Assim, pode-se afirmar que quanto mais precisas às informações que são utilizadas como suporte à decisão, mais acertadas são as decisões.

Dentro desta problemática, um aspecto que não pode ser desconsiderado é a capacidade de um SDI em manter seu conjunto de informações atualizadas. É desejável, portanto, que essas atualizações sejam feitas de forma automática, deixando a cargo do administrador apenas o refinamento e a introdução de regras específicas para o seu domínio.

Essas atualizações podem ser feitas através da obtenção de informações de centros especializados, os CSIRTs (*Computer Security Incident Response Team*), ou através da interação com outros SDIs.

De acordo com o CIDF (*Common Intrusion Detection Framework*) [14,58], a habilidade dos SDIs de compartilhar informações sobre incidentes de segurança é muito importante e permitiria que um sistema alertasse outro a respeito de ataques iminentes. No entanto, não seria seguro permitir o acesso direto aos componentes de um SDI por uma entidade externa. Assim, a introdução de um Módulo Gerenciador de Informações que gerencie este compartilhamento elimina o acesso direto aos componentes internos de um SDI e facilita o uso de funções de segurança necessárias a este tipo de aplicação como criptografia e assinatura digital [14].

O gerenciamento e a integração das bases de dados pode ser adicionado aos SDIs em forma de um novo módulo. Assim que este módulo for

ativado, passa a gerenciar o armazenamento, acesso, atualização e compartilhamento das informações utilizadas pelo SDI.

A Figura 4.1 ilustra de forma genérica um Módulo gerente de dados, atuando como o administrador das bases de dados de um SDI, interagindo com os demais componentes do SDI e com entidades externas ao SDI através de um *Web Service*.

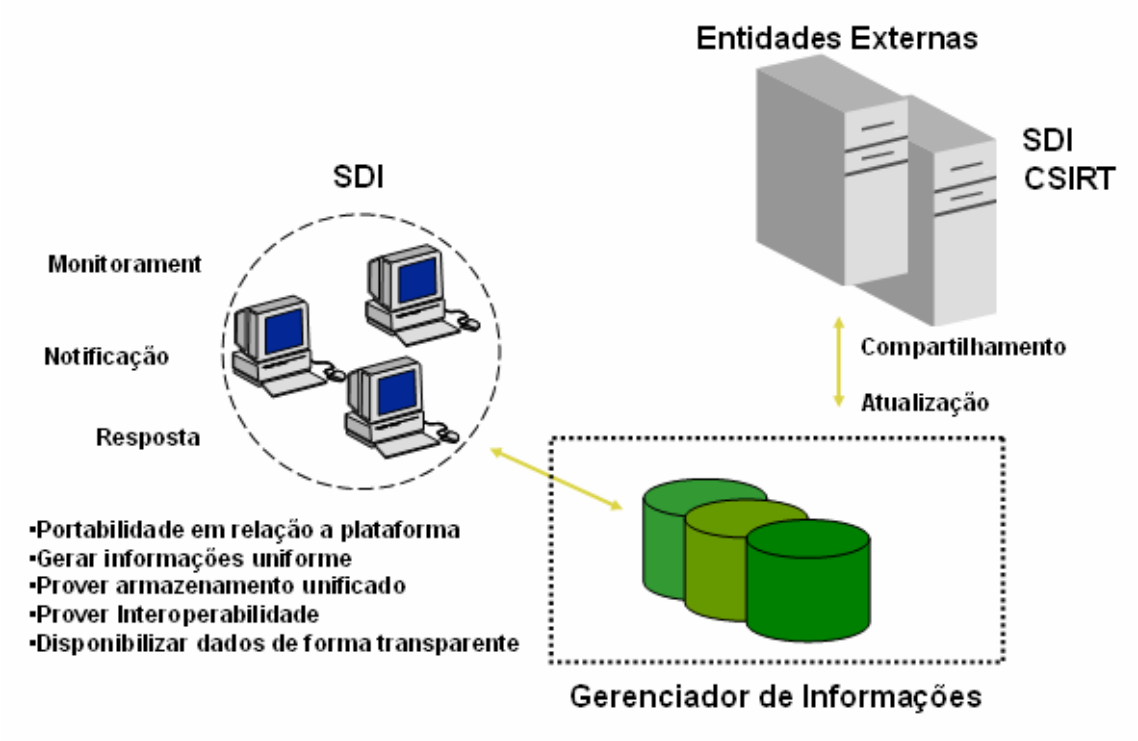

Figura 4.1. Visão Macro do Gerenciador de Informações.

Sendo assim, este Capítulo dedica-se a apresentação de uma proposta de Módulo Gerenciador de Informações [15,16]. Primeiro listam-se alguns requisitos que devem ser alcançados com a introdução do modelo proposto como gerente de dados. Em seguida discorre-se sobre a necessidade de gerenciamento de informações em SDIs. Depois, mostra-se a visão em camadas do modelo proposto detalhando o papel de cada camada que compõem o modelo. Por fim, descrevem-se os agentes e como é alcançado cada um dos requisitos listados.

4.1 Requisitos

Neste item, listam-se os requisitos que devem ser alcançados com a padronização das bases de dados em um SDI, conforme padronização proposta no Capítulo 3, bem como a introdução do Módulo Gerenciador de Informações responsável por manter estas bases de dados.

4.1.1 Requisitos para o serviço de armazenamento e gerenciamento

- Garantir portabilidade em relação à plataforma: a independência quanto à plataforma que os dados serão armazenados é um requisito necessário em se tratando de sistemas com as características de um SDI;
- Gerar informações uniformes: é importante lembrar que um SDI é um sistema complexo, composto muitas vezes por vários subsistemas, que geram e necessitam constantemente de informações [35]. Assim, torna-se imprescindível que todos os componentes deste organismo complexo gerem informações no mesmo formato facilitando o acesso e o gerenciamento desses dados;
- Prover armazenamento unificado: com esta característica objetiva-se aumentar a disponibilidade das informações, e diminuir o tempo de acesso. Estas características podem ser alcançadas através de mecanismos tais como:
    - Replicação: aumenta a disponibilidade de informações, reduzindo o risco de perda devido a existência de várias cópias das mesmas, além de otimizar o acesso a dados

read only e diminuir o tempo de acesso devido a possibilidade de buscar informações em pontos diferentes da rede, estando mais próximos ou com menor tráfego;

- Suprir a necessidade de interoperabilidade: esta capacidade de interagir com outros SDIs e com outras instituições que geram informações pertinentes aos SDIs é vital para execução eficiente das funções internas do SDI;
- Disponibilizar dados de forma transparente: além da capacidade de interoperabilidade, pode-se destacar também que esta interação deve acontecer de forma transparente. Como já mencionado um SDI é um sistema complexo, composto de vários subsistemas, os quais não devem se preocupar com o formato usado para armazenar os dados gerados internamente ou em como buscar informações geradas externamente ao SDI. Para tanto deve existir um subsistema que execute essas tarefas e forneça transparência de acesso a dados aos demais subsistemas.

4.2 Arquitetura em Camadas do Modelo Proposto

Nesta seção descreve-se o modelo proposto [15,16] baseado em agentes para gerenciamento de informações; capaz de fornecer flexibilidade e inteligência aos SDIs no que se refere à capacidade de manter suas informações atualizadas e seguras.

Com isso, objetiva-se aumentar a disponibilidade de informações, diminuir o tempo de acesso, reduzir o tempo para tomada de decisões,

melhorar a capacidade do sistema de agir de forma autônoma e diminuir a possibilidade de erros.

O gerenciamento e a integração das bases de dados podem ser adicionados a um SDI na forma de um novo módulo. Assim que este módulo for ativado, passa a gerenciar o armazenamento, acesso, atualização e compartilhamento das informações utilizadas pelo SDI. Em se tratando de compartilhamento, o SDI integra-se ao domínio que interage e troca informações de segurança.

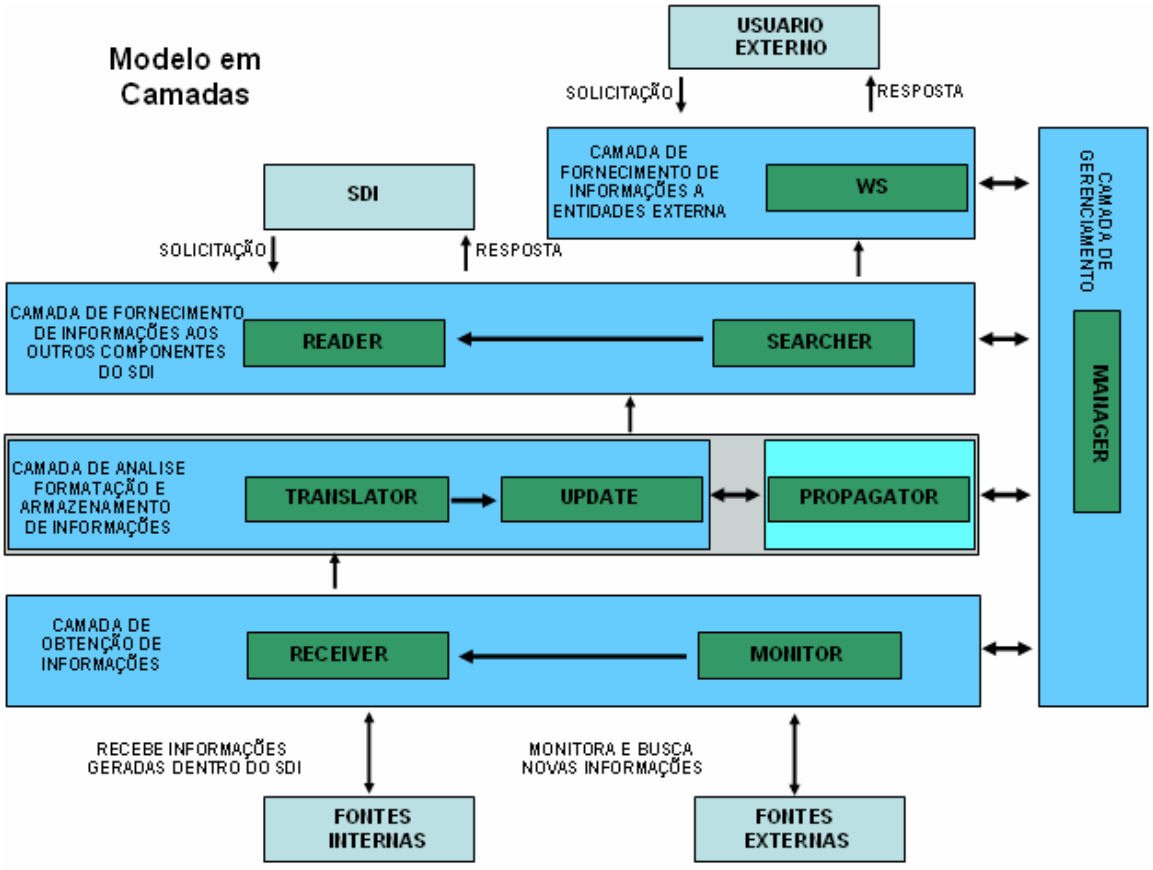

Figura 4.2. Visão em Camadas do Gerenciador de Informações.

O modelo proposto para o Módulo Gerenciador de Informações foi organizado em camadas às quais são descritas a seguir, de acordo com as funcionalidades a serem providas.

Na Figura 4.2 apresenta-se através de uma arquitetura organizada em camadas o Módulo Gerenciador de Informações proposto. Mostra-se ainda, na Figura 4.2, os agentes que compõem cada camada do modelo, e como acontece a interação para fornecimento e armazenamento de informações.

4.2.1 Camada de fornecimento de informações a entidades externas

Camada responsável por disponibilizar de forma segura a outros SDIs ou qualquer entidade externa, as informações geradas internamente ao SDI e/ou armazenadas em sua base de dados. É composta por agentes do tipo *WSAgent* que usa os serviços dos agentes, *SearcherAgent* e *ReaderAgent*, da camada inferior.

4.2.2 Camada de fornecimento de informações aos outros componentes do SDI

Como um SDI é um sistema complexo, composto de vários subsistemas especializados que executam tarefas específicas e de alta complexidade [35], faz-se desejável que, por exemplo, o componente responsável por executar uma ação de contra-ataque não tenha que se preocupar com o armazenamento de informações ou saber onde as informações necessárias para tomada de decisão estão localizadas, mantendo o foco em sua atividade específica.

Assim esta camada, que é composta por agentes do tipo *SearcherAgent* e *ReaderAgent*, é responsável por prover as informações corretas no formato correto aos demais componentes do SDI de forma transparente.

4.2.3 Camada de análise, formatação e armazenamento de informações

Camada que engloba: o tratamento e a análise de informações provenientes do meio externo; e formatação e armazenamento de informações geradas internamente e externamente. É responsável, ainda, por atividades como replicação no caso do SDI usar bases de dados replicadas.

Para tanto, constitui-se de agentes do tipo *UpdateAgent*, *TranslatorAgent* e *PropagatorAgent*. Esta camada é alimentada pelas informações recebidas da **camada de obtenção de informações** e alimenta as camadas responsáveis pelo fornecimento de informações.

4.2.4 Camada de obtenção de informações

Composta por agentes do tipo *ReceiverAgent* e *MonitorAgent*, faz a ligação do sistema com as fontes de informação da Internet, capturando novos dados, além de prover a abstração necessária aos demais componentes do SDI para que as informações geradas internamente sejam armazenadas em meio permanente de forma transparente.

4.2.5 Camada de gerenciamento

Responsável por monitorar a sociedade que compõe o Módulo Gerenciador de Informações, visando:

- Evitar falhas caso algum agente venha a ser finalizado de forma anormal. Para tanto, esta camada é capaz de reiniciar qualquer agente que tenha sua execução finalizada;
- Coletar informações da sociedade, e permitir que várias instâncias de um mesmo agente possam ser criadas, pelo

administrador, evitando a degradação do desempenho do Módulo Gerenciador;

- Iniciar o processo de replicação, a partir da criação de agentes *PropagatorAgent*, em intervalos previamente estabelecidos pelo administrador do Sistema;
- Permitir o monitoramento da "sociedade" por parte do administrador.

Esta camada é composta por agentes do tipo *ManagerAgent*.

4.3 Descrição dos Agentes

Cada uma das camadas, descritas previamente, é composta por um ou mais agentes que cooperam entre si para prover as funcionalidades pertinentes a cada camada. A seguir, detalham-se os agentes.

4.3.1 SeacherAgent

Agente responsável por atender solicitações para prover dados ao meio interno ou externo. Qualquer necessidade de informação gera uma requisição ao *SeacherAgent* que por sua vez cria agentes do tipo *ReaderAgent* para atender aos clientes.

Quando o agente *SeacherAgent* recebe uma nova requisição ele cria um agente *ReaderAgent* e passa para este o pedido a ser atendido e o cliente a ser servido. Deste ponto em diante todas as requisições daquele cliente serão encaminhadas diretamente ao *ReaderAgent* criado para atendê-lo. Quando o cliente não tiver mais necessidades de informação o *ReaderAgent* é destruído.

Desta forma, além da incumbência de receber toda e qualquer solicitação por informações, o *SeacherAgent* também deve monitorar e controlar os agentes do tipo *ReaderAgent*, ou seja, sua criação, seu comportamento e determinar o momento de sua destruição.

4.3.2 ReaderAgent

Este tipo agente é criado sob demanda pelo *SeacherAgent* para atender solicitações de um único e determinado cliente que precisa de informações. Por tanto ele não é um agente persistente tendo um ciclo de vida curto, existindo enquanto o cliente tenha novas solicitações.

É responsável pela extração da informação do meio onde está armazenada e pela geração de uma requisição ao *TranslatorAgent*, que converte do formato usado no armazenamento para o formato esperado pelo cliente, daí então procede à entrega dos dados ao cliente.

4.3.3 WSAgent

Usa os serviços dos agentes *SearcherAgent* e *ReaderAgent* para prover informações ao meio externo, ou seja, alimenta um *Web Service*s usado para disponibilizar as informações armazenadas internamente a outros SDIs ou a qualquer outra entidade externa.

4.3.4 TranslatorAgent

O *TranslatorAgent* tem fundamental importância dentro do Módulo Gerenciador de Informações, pois executa as transformações no formato das informações. Assim os demais subsistemas do SDI sempre vão obter dados prontos para serem consumidos.

Atende a solicitações de dois tipos de agentes que são: *ReaderAgente* para converter do formato usado no armazenamento para o formato usado pela entidade solicitante; e *UpdateAgent* para converter do formato que a informação foi gerada para o formato a ser armazenado.

4.3.5 ReceiverAgent

Desempenha um trabalho semelhante ao *SeacherAgent*, mas com a informação transitando no sentido inverso. Possui a responsabilidade de atender as requisições por armazenamento de informações, que podem ser oriundas do meio interno ou externo.

Assim como o *SeacherAgent* cria agentes do tipo *ReaderAgent*, o *ReceiverAgent* cria sob demanda agentes do tipo *UpdateAgent*. O monitoramento e gerenciamento dos agentes do tipo *UpdateAgent* criados também é de responsabilidade do *ReceiverAgent*.

4.3.6 UpdateAgent

Criado sob demanda pelos agentes do tipo *ReceiverAgent*, o *UpdateAgent* é responsável por escrever no meio persistente.

Quando criado recebe a informação a ser armazenada e verifica se já está no formato adequado, se não gera uma requisição ao *TranslatorAgent* que executa a conversão. Em seguida armazena o dado e retorna uma mensagem de sucesso à entidade que solicitou a escrita.

Todas as operações que alterem o estado das informações (atualizações, inserções, exclusões) são executadas pelo *UpdateAgent*.

4.3.7 MonitorAgent

Enviando constantemente requisições em busca de novos dados, o *MonitorAgent* agirá sempre que um novo dado for disponibilizado, recuperando-o e enviando-o ao *ReceiverAgent*.

O *MonitorAgent* é um agente do tipo reativo, ou seja, age em função das mudanças na fonte de informação.

É de extrema importância que essas fontes sejam reconhecidamente confiáveis, pois a partir de suas informações serão construídos novos mecanismos de reconhecimento de assinaturas de ataques e compostas as ações a serem executadas em reações a possíveis intrusões.

4.3.8 PropagatorAgent

O *PropagatorAgent* é responsável pela propagação de informações para outras bases de dados do domínio.

É criado em períodos de tempo pré-definidos pelo *ManagerAgent* e recebe um *log* de atualizações que devem ser propagadas, assim como também uma lista contendo todas as bases que devem ser atualizadas.

Os dados propagados são recebidos na base destino pelo *ReceiverAgent* que procede normalmente como a qualquer outra requisição para armazenamento.

Este tipo de agente pode não existir, dependendo das características do ambiente no qual o SDI é executado. Em ambientes replicados deve existir uma sociedade de agentes para gerenciar cada base de dados específica.

4.3.9 ManagerAgent

Este agente é responsável por monitorar a sociedade, recriar agentes permanentes em casos de falha e iniciar o processo de replicação

criando agentes do tipo *PropagatorAgent* sempre que houver necessidade de replicar informações.

Deve ser monitorado pelo administrador do sistema, pois tem papel fundamental no funcionamento adequando e ininterrupto da sociedade de agentes que compõe o Módulo Gerenciador de Informações.

4.4 Realização dos Requisitos

Na seção 4.1 deste Capítulo, expôs-se alguns requisitos para o Módulo Gerenciador de Dados, após a apresentação do modelo proposto, descreve-se a seguir como cada um dos requisitos é alcançado através do mesmo.

4.4.1 Portabilidade

Este requisito foi alcançado através da tecnologia XML[66], usada para definir o formato de armazenamento das informações utilizadas pelo SDI.

O formato XML fornece a flexibilidade desejada a fim de alcançar a portabilidade e adaptabilidade necessária para o domínio em questão [65]. Informações nesse formato podem ser migradas facilmente para outras plataformas como arquivos do sistema operacional ou para outros SGBDs (Sistemas Gerenciadores de Banco de Dados) com pouco esforço. Além de ser completamente compatível com *Web Services* permitindo interoperabilidade de maneira fácil e prática.

4.4.2 Geração de Informações Uniformes

Devido à necessidade de obter dados externos e de compartilhar as informações geradas internamente, torna-se vital a adoção de um formato único para geração de informações no SDI. Como os SDIs podem ter

arquiteturas diferentes, formas diferentes de trabalhar e geram informações em formatos diferentes, o Gerenciador de Informações através do *TranslatorAgent* garante esta característica sem gerar maiores impactos ou necessidades de mudanças na implementação de outros componentes do SDI.

4.4.3 Armazenamento Unificado

Sem a presença de uma entidade responsável pelo gerenciamento dos dados usados por um SDI, cada componente deste deve se preocupar com a obtenção de informação e também com o armazenamento. Como existem informações que são utilizadas apenas por um componente do SDI, ele não se preocupa em armazenar estes dados em formatos que possam ser manipulados por outras entidades internas ou externas, além de gerar desordem e dificultar a recuperação de informações.

Com o acesso e o armazenamento de dados sendo administrados pelo Gerenciador de Informações, torna-se possível o armazenamento unificado, conseqüentemente importantes características são inseridas no ambiente, como:

- Alta disponibilidade de dados: com as informações sendo armazenadas por uma única entidade e em principio em um único ponto, é possível implementar políticas de replicação aumentando a disponibilidade dos dados;
- Política de segurança dos dados armazenados: a introdução de repositórios replicados em si já garante certo grau de segurança dos dados em caso de perda de um nó da rede, por exemplo. Além disso, a introdução de políticas de *backup* torna-se possível em um ambiente com armazenamento unificado;

- Política de segurança de acesso aos dados: o uso de artifícios de segurança, como autenticação e criptografia, é facilitado com a introdução do Gerenciador de Informações, pois as entidades que podem acessar as bases de dados são os agentes de *software* deste módulo, podendo ser inseridos nos demais componentes do SDI a capacidade de se autenticar junto ao Gerenciador de Informações.

4.4.4 Acesso Transparente

Acesso transparente é a habilidade de um componente tem de requisitar informações e recebê-las no formato requisitado sem nenhuma preocupação com o local onde ela esta armazenada ou com o formato usado para gravá-la em meio persistente.

O Gerenciador de Informações provê esta funcionalidade através dos agentes de extração de dados, *SearcherAgent* e *ReaderAgent*, e do agente de conversão de formatos *TranslatorAgent*.

4.4.5 Interoperabilidade

Para atender este requisito vários agentes do Módulo Gerenciador de Informações são envolvidos, além do uso de um *Web Service.* Dentre os agentes que compõem a sociedade podemos destacar como fundamentais para garantir a interoperabilidade os agentes *WSAgent* e *MonitorAgent*, os quais foram concebidos especificamente para este objetivo.

A Figura 4.3 ilustra a obtenção e disponibilização de informações por parte de um SDI que utiliza os serviços do Módulo Gerenciador de Informações. A obtenção de dados externos é feita através do *MonitorAgent* que é capaz de gerar requisições a um *Web Service* mantido por uma entidade

externa como um CSIRT ou um outro SDI. Já a disponibilização das informações internas é feita através do *WSAgent* e de um *Web Service* mantido pelo Módulo Gerenciador de Informações.

Em [47] desenvolveu-se e integrou-se ao NIDIA um *Web Service*. Neste trabalho adota-se o *Web Service* já desenvolvido em [47] para o estabelecimento de comunicação com entidades externas.

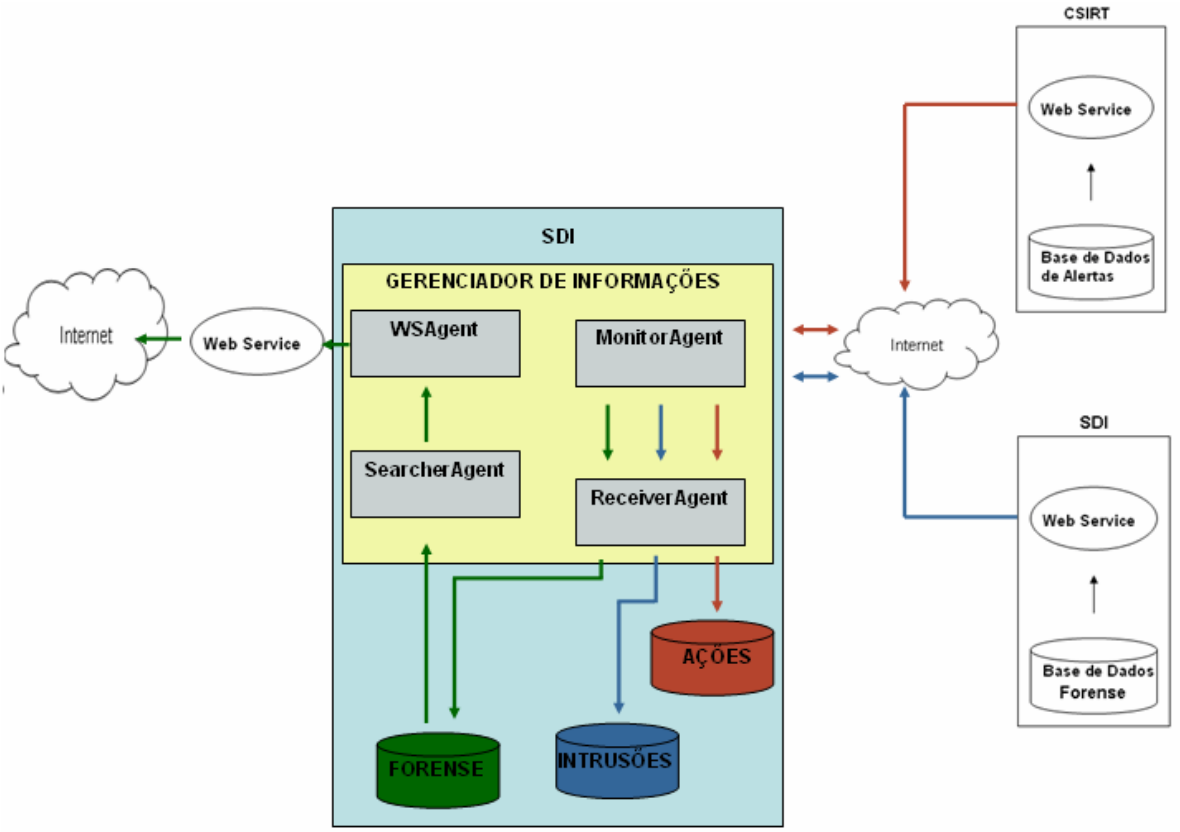

Figura 4.3. Obtenção e Disponibilização de Informações por um SDI Através do Módulo Gerenciador de Informações.

Este requisito é necessário, pois objetiva possibilitar que o SDI atualize de forma automática, primeiramente seu conjunto de dados de ações de resposta e assinaturas de intrusões com base em medidas sugeridas nos alertas de segurança emitidos pelos CSIRTs, e depois trocar informações forenses geradas pelos SDIs.

O compartilhamento de informações, então, dar-se-á em dois níveis: primeiro, entre CSIRT e SDI; e segundo de SDI para SDI. Nas subseções seguintes, descrevem-se cada um destes níveis de interação.

4.4.6 Interação entre SDI e CSIRT

O compartilhamento de informações neste nível tem como objetivo possibilitar que os SDIs atualizem de forma automática principalmente seus dados sobre ações de resposta. Caso o CSIRT possua um sistema automatizado para recebimento de notificações, elas seriam prontamente processadas e suas informações disponibilizadas, também de forma automática, a todos os órgãos e entidades interessados em tais informações de segurança.

Através de um *Web Services*, o *MonitorAgent* pode enviar uma requisição buscando por novos alertas disponibilizados pelos CSIRTs. O serviço recebe a requisição, a processa e devolve a resposta. Sempre que o *MonitorAgent* receber um novo alerta emitido por um CSIRT, será repassado ao *ReceiverAgent* que se encarrega juntamente com os agentes *UpdateAgent* e *TranslatorAgent* de tratar e armazenar as informações.

Para que o compartilhamento de informações entre os SDIs e os CSIRTs possa ser implementado é preciso que alguns requisitos sejam atendidos [3]:

- Estabelecer e concordar sobre um conjunto de elementos que irão utilizar e o que estes elementos significam;
- A existência de um serviço que permita a comunicação entre diferentes sistemas. Este requisito pode ser plenamente satisfeito através da tecnologia de *Web Services*.

Além dos requisitos listados, é necessário que os alertas de segurança emitidos pelos CSIRTs sejam padronizados e codificados de forma a permitir o seu processamento automático. Com a padronização evita-se que o SDI tenha que entender e decodificar vários formatos.

Em [48] propõe-se um formato de codificação de alertas emitidos pelos CSIRTs, utilizando XML, como uma extensão do CAP (*Common Alerting Protocol*) [11]. Neste trabalho adota-se o formato proposto em [48] para a troca de informações entre CSIRTs e SDIs.

4.4.7 Interação entre SDIs

Neste nível acontece principalmente a troca de informações forenses. Dados forenses são registros de danos causados por ataques bem-sucedidos e/ou tentativas de ataques, que podem ser utilizados na identificação de tentativas de intrusões provenientes de uma mesma origem ou domínio, ou ainda serem utilizadas em investigações futuras.

Essas informações podem, ainda, serem usadas para avaliar o nível de segurança do domínio, a eficiência das ferramentas usadas para prover esta segurança e a eficiência do próprio SDI.

Comparar informações geradas por vários SDIs e analisar tipos de ataques sofridos, ações de respostas executadas, estratégias de ações adotadas, com certeza resultará em um refinamento das ações de segurança seguidas pelos administradores e proporcionará ambientes mais seguros.

Para esta comunicação, um *Web Services* deve ser mantido pelo SDI. Esse serviço é provido pelo Gerenciador de Informações, e a entidade responsável por enviar requisições ao outro SDI é o *MonitorAgent*. Essas requisições podem referir-se a todas as informações forenses do SDI destino

da requisição, assim como informações sobre eventos específicos. O *Web Service* recebe a requisição, analisa-a e retorna as informações solicitadas. Os agentes envolvidos no processo de atendimento das solicitações são: *WSAgent* que recebe a requisição e a repassa ao *SeacherAgente*, responsável por criar e manter um *ReaderAgent* que faz a extração do dado do meio físico e entrega ao *WSAgent*, que por fim procede a entrega ao *MonitorAgent* do SDI solicitante.

Para que o compartilhamento de informações entre os SDIs possa ser implementado é necessário apenas que o Gerenciador de Informações esteja presente nos SDIs que desejam se comunicar.

A troca de informações é feita através da utilização de XML e *Web Services*, e da mesma forma que a comunicação entre SDI e CSIRT é necessária à adoção de um formato padronizado. Na seção 3.6.1 do Capítulo 3 foi proposto um modelo para troca de informações forenses. O modelo apresentado foi desenvolvido em XML, a partir do padrão CAP.

4.5 Conclusões

Neste Capítulo discorreu-se sobre a necessidade de gerenciamento de informações em SDIs. Esse gerenciamento envolve a administração e exportação de informações geradas internamente e a importação de informações geradas externamente ao SDI. Contudo, viu-se que não é seguro permitir o acesso direto aos componentes de um SDI por entidades externas.

A introdução de um Módulo Gerenciador de Informações que se responsabilize por este compartilhamento eliminaria o acesso direto, e facilitaria o uso de funções de segurança necessárias a este tipo de aplicação, como criptografia e assinatura digital. Além de inserir ao ambiente soluções

para atender aos requisitos de: Portabilidade dos Dados, Geração de Informações Uniformes, Armazenamento Unificado, Acesso Transparente a Dados e Interoperabilidade.

Propôs-se então o Módulo Gerenciador de Informações, apresentado através de uma arquitetura em camadas. Assim detalhou-se sua arquitetura e seu funcionamento a partir do detalhamento do papel de cada camada que compõem o modelo e de cada agente que compõem cada camada.

Por fim, mostrou-se como cada um dos requisitos listados é alcançado a partir do Módulo Gerenciador de Informações proposto.

5 IMPLEMENTAÇÃO

O modelo apresentado para o gerenciamento de informações constitui-se de uma proposta genérica de mecanismo de atualização e gerenciamento de dados para SDIs (Sistemas de Detecção de Intrusão). Neste contexto, descreve-se neste Capítulo o estado da implementação.

5.1 Ferramentas Utilizadas

Como se mencionou, o Gerenciador de Informações foi concebido como um Sistema Multiagentes combinado com a tecnologia de *Web Service*. Como ferramentas para implementação utilizou-se PASSI, JADE e Xindice.

5.1.1 PASSI

PASSI (*Process For Agent Societies Specification and Implementation*) [17,46] foi utilizado como metodologia de desenvolvimento da sociedade de agentes, pois integra os modelos de projeto e conceitos orientados a objetos de engenharia de software e inteligência artificial usando notação UML (*Unified Modeling Language*).

Esta metodologia, PASSI, é composta de alguns modelos relativos a diferentes níveis do projeto, e vários passos que orientam o processo de construção de Sistemas Multiagentes. PASSI adota UML como linguagem de modelagem, por esta ser aceita amplamente em ambientes acadêmicos e industriais, evitando impactos na produtividade decorrente da adoção de uma linguagem completamente nova [46].

Dentre as fases estabelecidas pela metodologia PASSI para construção de Sistemas Multiagentes e seus respectivos produtos, pode-se destacar [46]:

- Descrição do Domínio (*Domain Description*): uma descrição funcional do sistema usando diagrama convencional, UML, de casos de uso;
- Identificação de Agentes (*Agent Identification*): responsabilidades são separadas por agentes, sendo que os agentes são representados como pacotes em notação UML. Um diagrama de pacotes representando todo o sistema ou parte deste é o resultado desta fase;
- Identificação de Papéis (*Role Identification*): diagramas de seqüência são usados para explorar as responsabilidades ou papéis de cada agente através de cenários específicos;
- Especificação de Tarefas (*Task Specification*): a capacidade de cada agente é especificada através de diagramas de atividades;
- Descrição de Ontologia (*Ontology Description*): uso de diagramas de classe para descrever o conhecimento individual designado aos agentes e às interações entre eles.

5.1.2 JADE

Como plataforma para implementação do sistema multiagentes utilizou-se JADE (*Java Agent Development Framework*) [31]. A escolha é devida ao fato deste ser um framework, totalmente implementado em Java de acordo as especificações FIPA [26], que simplifica o desenvolvimento de sistemas multiagentes.

Além disso, JADE é um *software* livre e distribuído, por Telecom Itália [61], em código aberto sob os termos da LGPL (*Lesser General Public License)* Versão 2.

5.1.3 Xindice

Com a finalidade de garantir maior portabilidade dos dados armazenados e permitir o menor esforço possível na extração e entrega das informações, optou-se por utilizar um SGDB XML nativo. O Xindice [62] é um *software* livre e em código aberto, que permite dentre outras coisas:

- O armazenamento nativo em XML;
- A capacidade de buscar arquivos completos ou partes de arquivos de forma transparente;
- A total compatibilidade com o formato adotado para armazenamento e troca de dados com entidades externas;
- O gerenciamento dos dados de forma eficiente;
- O acesso direto aos dados através de ferramenta de linha de comandos;
- A realização de *backup* e recuperação através de utilitários do próprio Xindice.

Para sistemas desenvolvidos em Java, Xindice prover uma implementação da API (*Application Programming Interface*) XML:DB. Através desta API a portabilidade de aplicações que utilizam banco de dados XML é semelhante a alcançada com as que usam JDBC para acessar bancos de dados relacional.

5.2 Construção do Sistema Multiagentes

Após a concepção do Gerenciador de Informações e a escolha da metodologia a ser utilizada para construí-lo, no caso PASSI, passou-se à fase de desenvolvimento. Nas subseções seguintes apresentam-se a definição dos

agentes, a definição da ontologia da sociedade de agentes e as responsabilidades e tarefas de cada agente. Em seguida, na seção 5.3 descreve-se a fase de codificação, sobre a plataforma JADE, do Sistema Multiagentes.

5.2.1 Definição dos Agentes

O primeiro diagrama gerado, conforme a metodologia adotada, PASSI, para desenvolvimento do Sistema Multiagentes, é o Diagrama de Descrição de Domínio, que contém uma descrição funcional do sistema usando diagrama convencional de casos de uso.

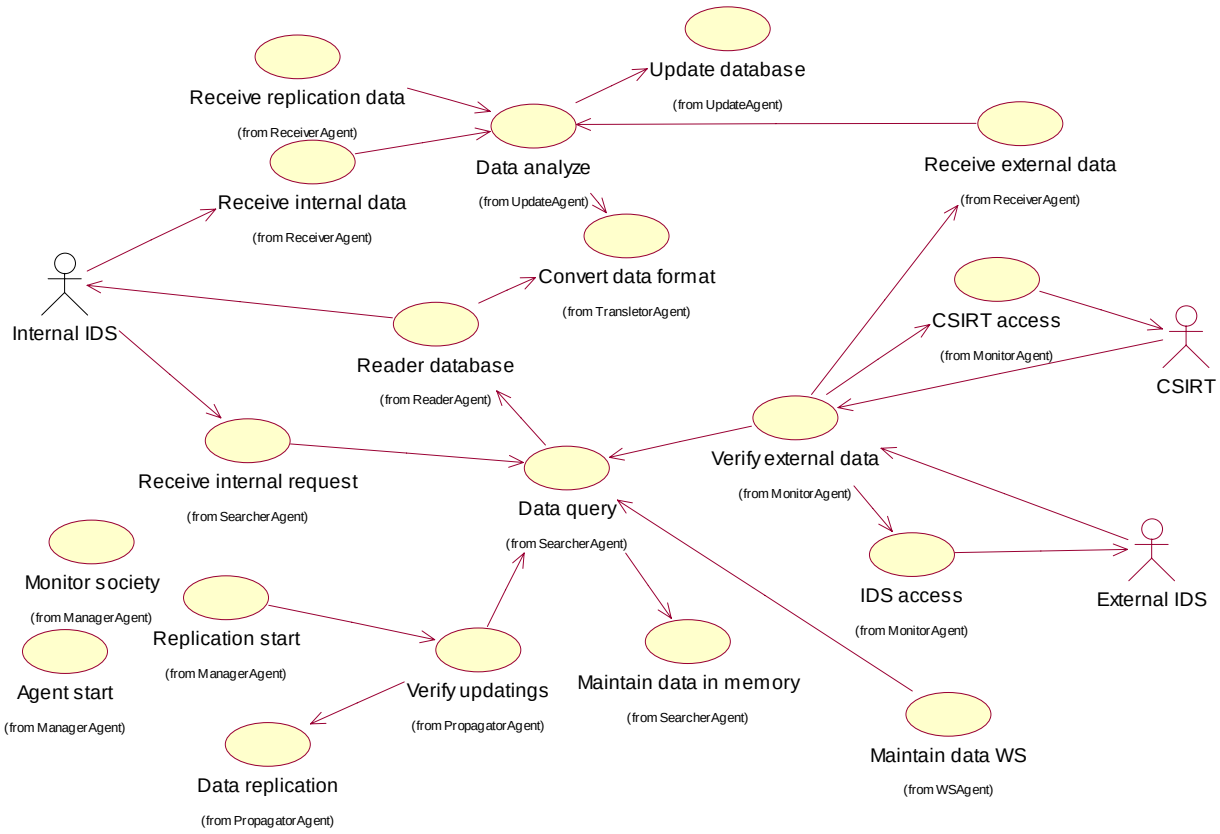

Figura 5.1. Descrição do Domínio a Partir de um Diagrama PASSI para Descrição de Domínio.

A figura 5.1 mostra a contrução do Diagrama de Descrição de Domínio. Ainda nesta figura, cada caso de uso representa uma funcionalidade

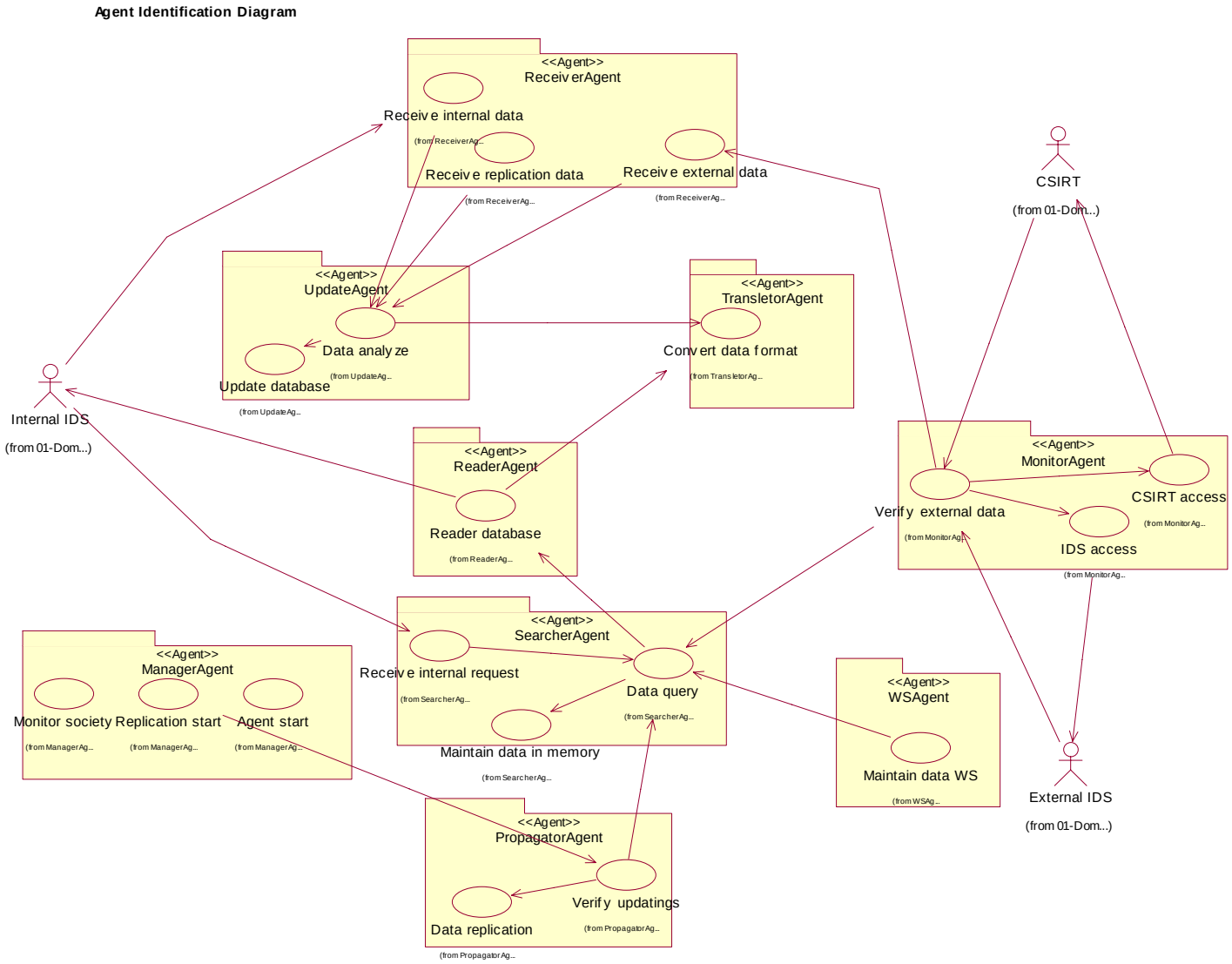

Figura 5.2. Identificação dos Agentes do Sistema Multiagentes a Partir de Diagrama PASSI Utilizado na Identificação de Agentes do Sistema.

do sistema. Os atores representam as entidades, componentes internos do SDI e entidades externas, que interagem com o Sistema Multiagentes em contrução.

Assim descrevem-se os requisitos em termos de diagramas de caso de uso. Como resultado, a fase de descrição de domínio é uma descrição funcional do sistema composta de diagramas de caso de uso usando notação UML.

A Figura 5.2 ilustra o Diagrama PASSI utilizado na Identificação de Agentes. É um diagrama de pacotes UML gerado a partir do Diagrama de Descrição de Domínio. Mostra-se de forma macro, na Figura 5.2, como os agentes do Gerenciador de Informações se relacionam e suas atribuições principais. Neste diagrama um agente é apresentado como um pacote que possui um ou mais casos de uso representando suas responsabilidades.

5.2.2 Definição da Ontologia

Para cada comunicação entre agentes é preciso especificar três elementos: ontologia, linguagem e protocolo de interação. Enquanto várias

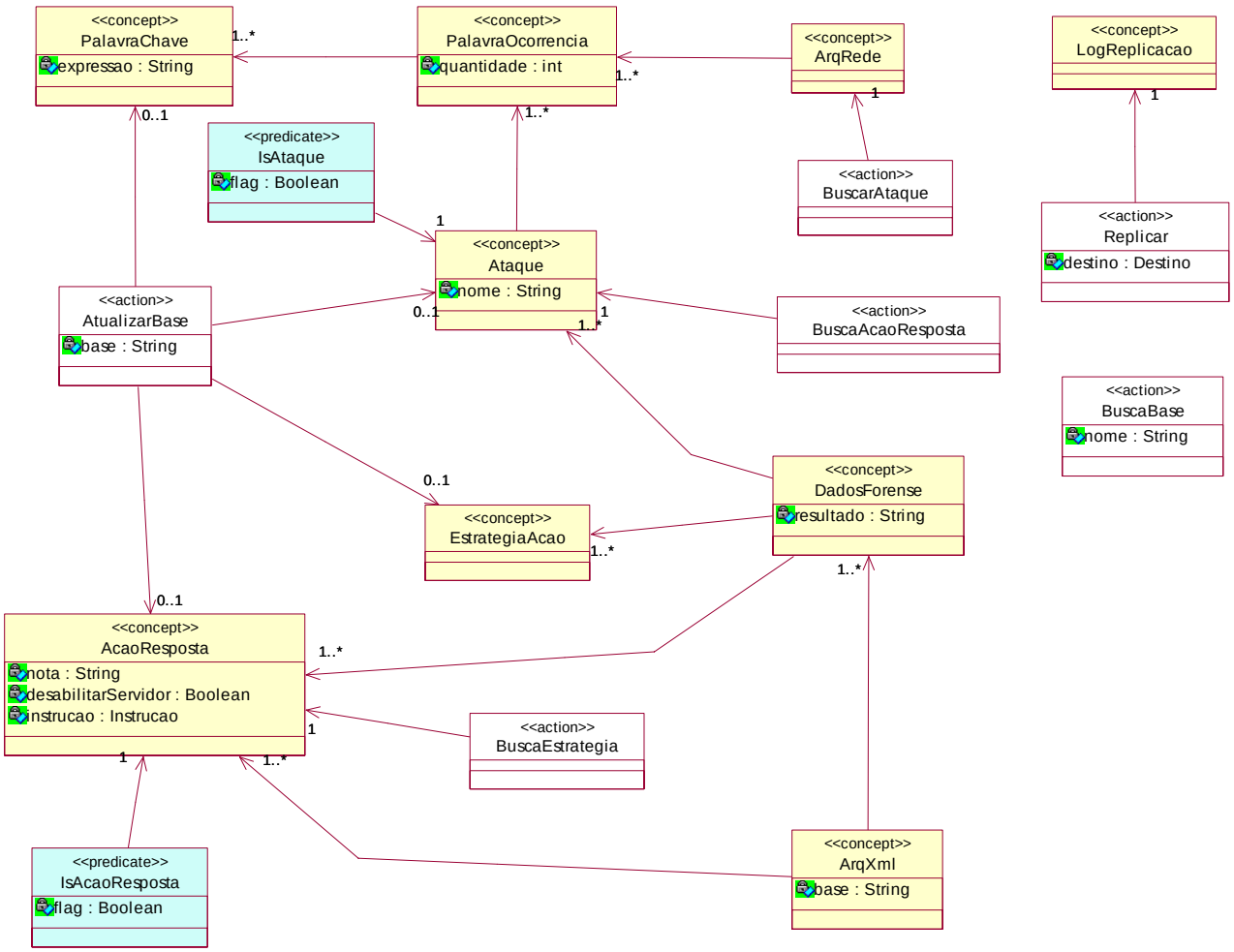

Figura 5.3. Ontologia do Sistema Multiagentes Ilustrada Através de Diagrama PASSI de Descrição da Ontologia do Domínio.

linguagens e protocolos de interação são padronizados pela FIPA, a ontologia, que é relacionada freqüentemente ao problema foi definida neste trabalho como conseqüência da aplicação específica.

Para detalhar a ontologia concebida para compor a solução, usou-se um diagrama de classes chamado Diagrama de Descrição de Ontologia, conforme Figura 5.3. Neste Diagrama descreve-se a ontologia do domínio, onde cada classe representa uma entidade.

A Figura 5.3 mostra a ontologia utilizada para implementar o Gerenciador de Informações. Vale lembrar que esta ontologia foi desenvolvida já para suportar a integração do Gerenciador de Informações ao NIDIA (*Network Intrusion Detection System Based on Intelligent Agents*), assim, alguns dos elementos só existem, e fazem sentido, por causa do NIDIA. A ontologia é composta de conceitos, ações e predicados utilizados pelo domínio, os quais são descritos abaixo.

- **Conceitos:** representados no diagrama pelas classes em amarelo, um conceito representa uma entidade chave dentro do domínio que possui uma estrutura complexa e que pode ser definida em termos de atributos [32]. A seguir descreve-se cada um dos conceitos apresentados na Figura 5.3:
    - PalavraChave: expressão usada para determinar a ocorrência de ataques. Dentro do NIDIA compõe a base de palavras chaves otimizada OKDB (*Optimized KeyWord Data Base*);
    - PalavraOcorrencia: agrega o conceito de *PalavraChave* mais o número de ocorrência dessa palavra em um

determinado arquivo analisado pelo componente responsável em detectar intrusões. No NIDIA o SMA (*System Monitoring Agent*), agentes de monitoramento que compõem a camada de monitoramento, a utiliza na geração do *ArqRede*;

- ArqRede: gerado pelo SMA, contém um vetor de contagem de palavras para cada conexão de rede. É enviado ao SEA (*System Security Evaluation Agent*), agentes de avaliação de segurança do sistema que compõem a camada de análise, que depois de gerar o índice de severidade, o envia ao SCA (*System Controller Agent*), agentes controladores de ações que compõem a camada de reação. O SCA por sua vez utiliza esse arquivo para identificar a intrusão;
- Ataque: nomeado na ontologia como *Ataque*, este conceito deve identificar o evento que está sob investigação. Com o *ArqRede*, o SCA, entidade responsável em identificar e responder as intrusões no NIDIA, busca a identificação do *Ataque;*
- AcaoResposta: o SDI deve estar apto a responder a um evento de intrusão e não apenas identificá-lo. Assim, após a identificação do evento uma ação deve ser determinada como resposta, a esta reação denominou-se *AcaoResposta*. Para isso o SCA usa o *Ataque* identificado e busca a *AcaoResposta* a ser realizada;

- EstrategiaAcao: esse conceito define o procedimento que deve ser seguido para a execução da *AcaoResposta* determinada. A identificação dessa estratégia de ação é feita pelo SCA depois de determinar qual ação será realizada em represália ao evento identificado;
- DadosForense: representa o repositório de informações de *log* do SDI onde todas as informações geradas no decorrer do processo descrito anteriormente são armazenadas para posterior recuperação e análise;
- ArqXml: conceito que representa os dados recebidos das fontes externas para manter as informações do SDI atualizadas;
- LogReplicacao: este conceito identifica as informações que necessitam ser propagadas para as demais bases de dados do SDI.

- **Ações:** é um tipo especial de conceito que indica uma ação que pode ser executada por um agente. As ações são geralmente executadas em reação a uma mensagem recebida ou após a avaliação de um predicado [32]. Na ontologia definiram-se as seguintes ações:
  - BuscarBase: esta ação pode ser desencadeada para obtenção de uma das bases de dados, ou parte de uma, do SDI. É sempre realizada durante a fase de identificação do *Ataque*, na geração do *ArqRede*. Neste contexto, a

base de palavras-chaves otimizada deve ser retornada ao solicitante;

- BuscarAtaque: ação realizada durante a fase de identificação de *Ataque*. Contém um objeto do tipo *ArqRede* e deve retornar uma lista de *Ataques* ao solicitante;
- BuscarAcaoResposta: ação realizada após a identificação do *Ataque*. Contém um tipo *Ataque* e deve retornar uma lista contendo as possíveis *AcaoRespostas* ao solicitante;
- BuscaEstrategia: ação realizada após a identificação da *AcaoResposta*. Ela contém um tipo *AcaoResposta* e deve retornar uma lista contendo as possíveis *AcaoEstrategias* ao solicitante;
- AtualizarBase: Ação realizada sempre que alguma das bases de dados sofrer modificações. Contém o nome da base que sofrerá a ação, mais o conteúdo a ser anexado ou removido da base;
- Replicar: realizada somente pelo *ManagerAgent,* esta ação contém um tipo *LogReplicacao*, e é dispara sempre que houver necessidade de propagar dados para outras bases do sistema.

- **Predicados:** são expressões que dizem algo sobre o estado do domínio. Essas expressões são sempre avaliadas como verdadeiras ou falsas [32]. Os predicados apresentados na figura 5.3, são detalhados a seguir:

- isAtaque: usado para determinar se o conteúdo passado na ação de identificação de *AcaoResposta* identifica um *Ataque* no sistema;
- isAcaoResposta: predicado usado para determinar se o conteúdo passado na ação de identificação de *EstrategiaAcao* identifica uma *AcaoResposta* válida no sistema.

5.2.3 Descobrindo as Responsabilidades ou Papéis dos Agentes

A Identificação de responsabilidades ou papéis está baseada na exploração de todos os possíveis caminhos do Diagrama PASSI de Identificação de Agentes, Figura 5.1, que envolve a comunicação inter-agente. Cada comunicação descreve um cenário onde agentes interagem e trabalham para alcançar um comportamento exigido pelo sistema. Este comportamento é composto de várias relações de comunicação.

Estes cenários são extraídos por meio de diagramas de sequência, Diagramas PASSI de Identificação de Papéis, nos quais são usados objetos para simbolizar as responsabilidades. Cada agente pode participar em diferentes cenários com diferentes responsabilidades, assim como, em um mesmo cenário participar com responsabilidades distintas.

Nas Figuras 5.4, 5.5, 5.6. 5.7 e 5.8 mostram-se os principais cenários utilizados para descobrimento das responsabilidades dos agentes que compõem o Gerenciador de Informações. Cada objeto nos diagramas representa um papél e são nomeados da seguinte forma: *<nome_papel> : <nome_agente>*.

Inicialmente, mostra-se, na figura 5.4, o cenário de um atendimento de uma solicitação feita ao Gerenciador de Informações oriunda de um componente interno do SDI. Neste cenário o *SeacherAgent* possui as responsabilidades ou papéis de Atendedor, Verificador e Criador; o *ReaderAgent* por sua vez possui os papéis de Leitor e Entregador; e o *TranslatorAgent* que possui a responsabilidade de Conversor.

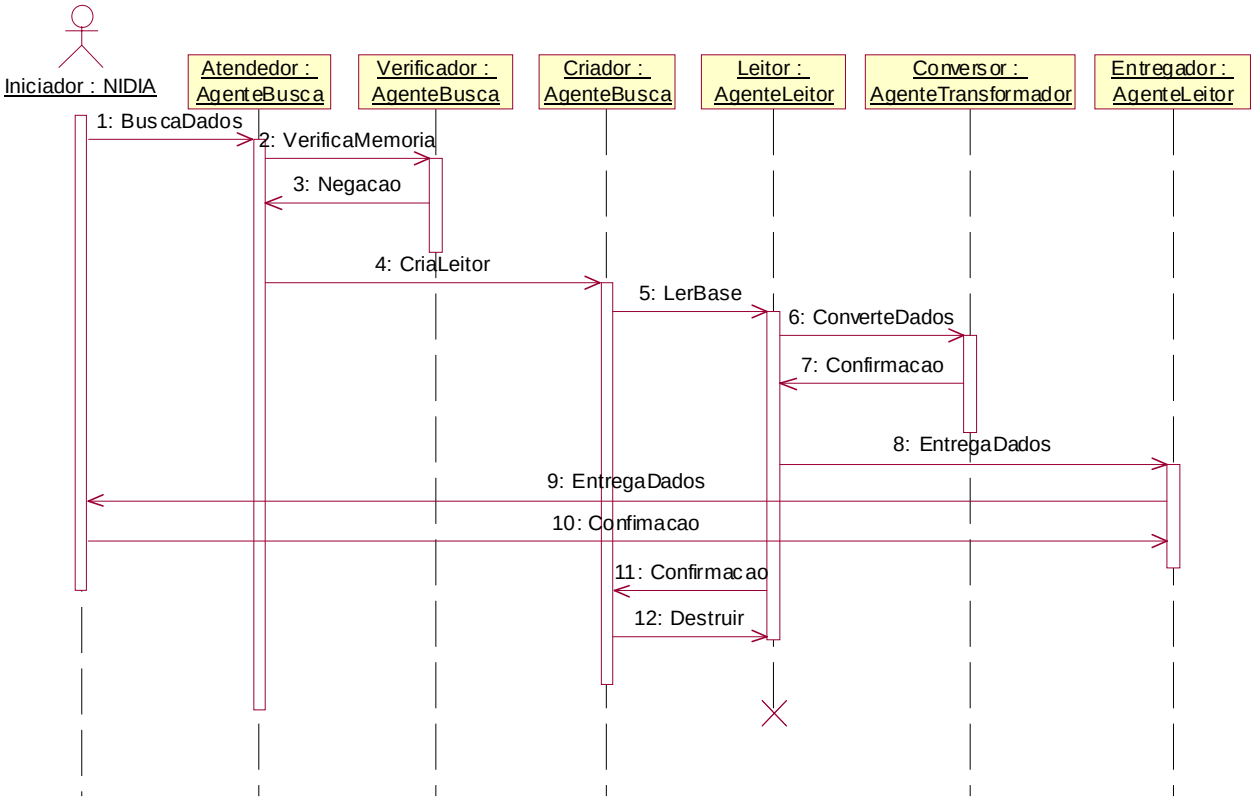

Figura 5.4. Identificação das Responsabilidades dos Agentes a Partir de Diagrama PASSI de Identificação de Responsabilidades. Cenário Atendimento de Solicitação Interna.

O diagrama de seqüência, Diagrama PASSI de Identificação de Papéis, descrito na figura 5.4 desenvolve-se da seguinte forma: (1) Um componente interno do SDI faz uma solicitação ao *SeacherAgent*, (2) que verifica a existência dos dados requeridos em memória, (3) não encontrando, (4) cria um *ReaderAgent* para buscar os dados em meio persistente; O *ReaderAgent* (5) ler a base, e em seguida (6,7) solicita ao *TranslatorAgent* a

conversão dos dados para o formato esperado pelo componente do SDI que solicitou as informações, logo após a conversão procede a (8,9) entrega dos dados, (10) recebe a confirmação que o dado foi entregue, (11) comunica ao *SeacherAgent* que não existem mais solicitações a serem atendidas; O *SeacherAgent* então ordena o fim da execução do *ReaderAgent*.

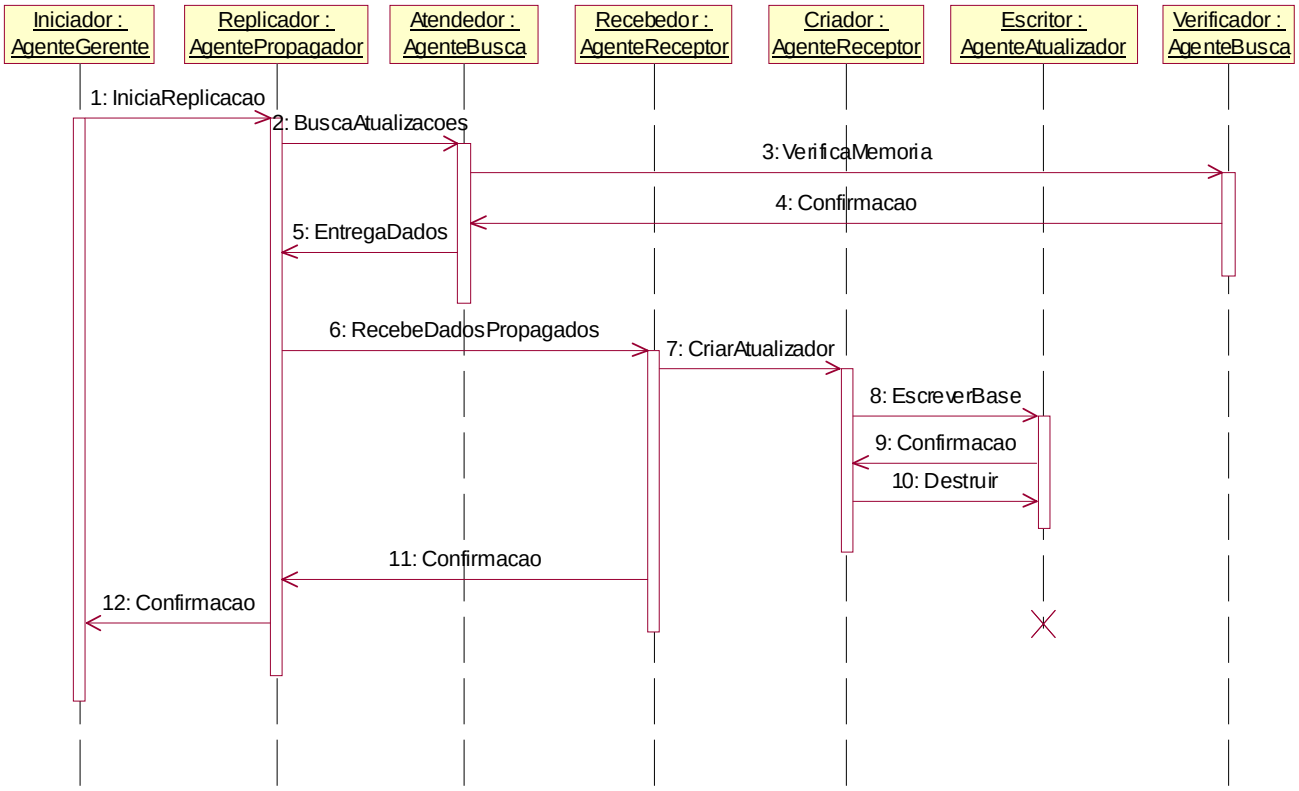

Figura 5.5. Identificação das Responsabilidades dos Agentes a Partir de Diagrama PASSI de Identificação de Responsabilidades. Cenário Replicação.

A replicação das informações é detalhada na Figura 5.5. Os agentes que participam deste evento são: *ManagerAgent* com as responsabilidades de Iniciador e Criador; *PropagatorAgent* com a responsabilidade de Replicador; *SeacherAgent* que neste cenário atua como Atendendor e Verificador; *ReceiverAgent* que possui as responsabilidades de Receptor e Criador; e com o papel de Escritor está o *UpdateAgent.* Para esta ilustração os dados a serem

replicados se encontram em memória, motivo pelo qual o *ReaderAgent* não foi incluído.

O cenário descrito na Figura 5.5 é iniciado pelo *ManagerAgent* (1) através da criação de um *PropagatorAgent*; (2) o agente propagador solicita ao *SeacherAgent* os dados a serem atualizados; o *SeacherAgent* (3) verifica se os dados solicitados estão em memória, (4) caso positivo, (5) procede a entrega ao *PropagatorAgent*; o *PropagatorAgent* então (6) replica os dados que são recebidos por agentes do tipo *ReceiverAgent* na base de destino; O *ReceiverAgent*, então (7) cria um *UpdateAgent*, que após executar a (8,9) gravação em meio persistente será (10) destruído, neste ponto o *ReceiverAgent* (11) confirma o recebimento ao agente responsável pela replicação; o agente Propagador (12) confirma o sucesso ao *ManagerAgent* e finaliza sua execução.

Um outro importante cenário é o armazenamento de dados gerados por componentes do SDI. Este evento, Figura 5.6, é explorado com intuito de descobrir novos papéis para os agentes. Após o detalhamento, têm-se que os agentes: *ReceiverAgent* com os papéis de Recebedor e Criador, *UpdateAgent* como Analizador e Escritor; e *TranslatorAgent* com a responsabilidade de Conversor, cooperam para realizar essa função.

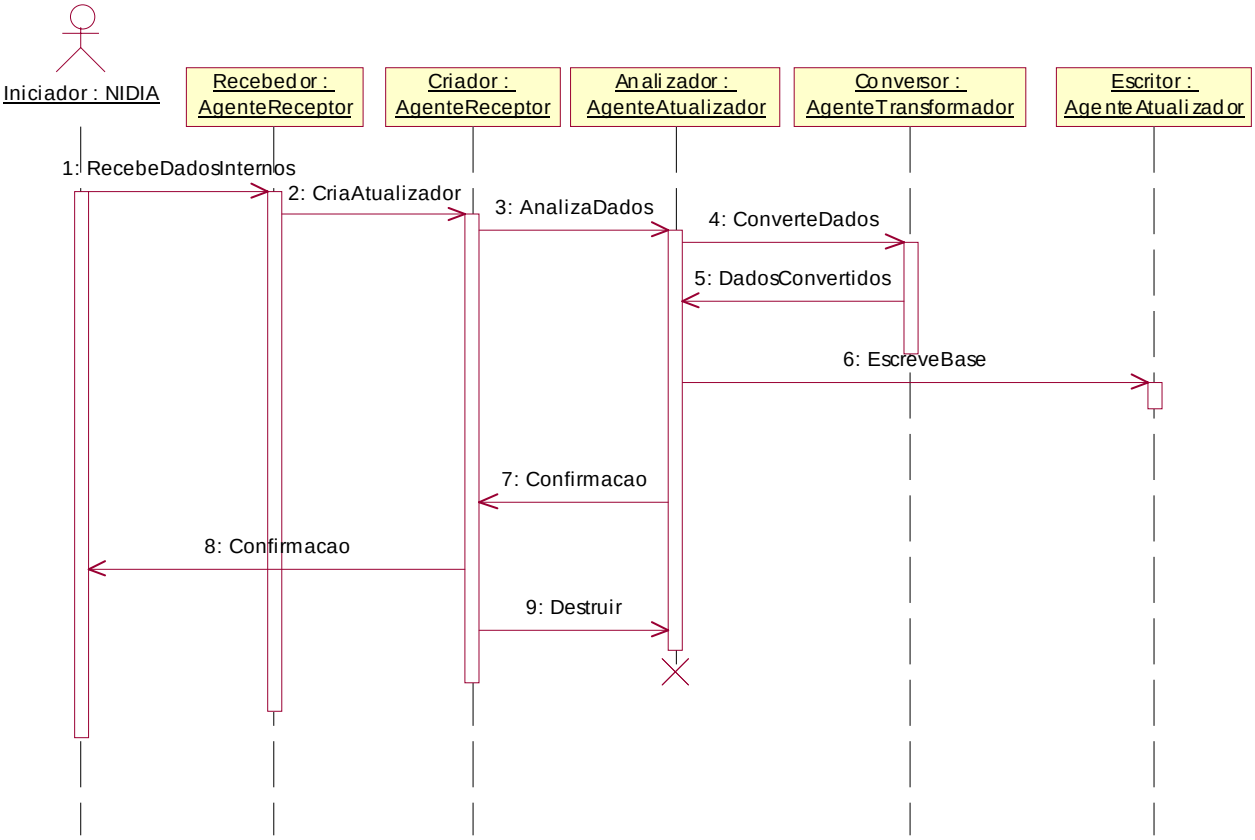

Figura 5.6. Identificação das Responsabilidades dos Agentes a Partir de Diagrama PASSI de Identificação de Responsabilidades. Cenário Armazenamento de Dados Internos.

O digrama de seqüência representado, na Figura 5.6, para ilustrar como ocorre uma solicitação por armazenamento de informações opera da seguinte forma: (1) o *ReceiverAgent* recebe de um dos componentes que compõem o SDI os dados a serem armazenados, (2) cria um *UpdateAgent*; o *UpdateAgent* por sua vez (3) procede a análise dos dados e constata que os mesmos precisam de conversão, (4) solicita a conversão ao *TranslatorAgent*, (5) recebe-os transformados e (6) armazena-os, daí o *UpdateAgent* devolve uma (7) confirmação ao *ReceiverAgent* que o criou; O *ReceiverAgent* (8) confirma a operação ao SDI e (9) envia uma mensagem para que o *UpdateAgent* se auto destrua.

As Figuras 5.7 e 5.8 ilustram os dois últimos cenários escolhidos para serem usados na identificação de responsabilidades.

O primeiro cenário, Figura 5.7, trata do armazenamento de dados provenientes de entidades externas, no qual aparecem os agentes *MonitorAgent*, *SearcherAgent*, *ReceiverAgent* e *UpdateAgent* com suas respectivas responsabilidades de Buscador, Atendedor e Verificador, Recebedor e Criador, e Escritor.

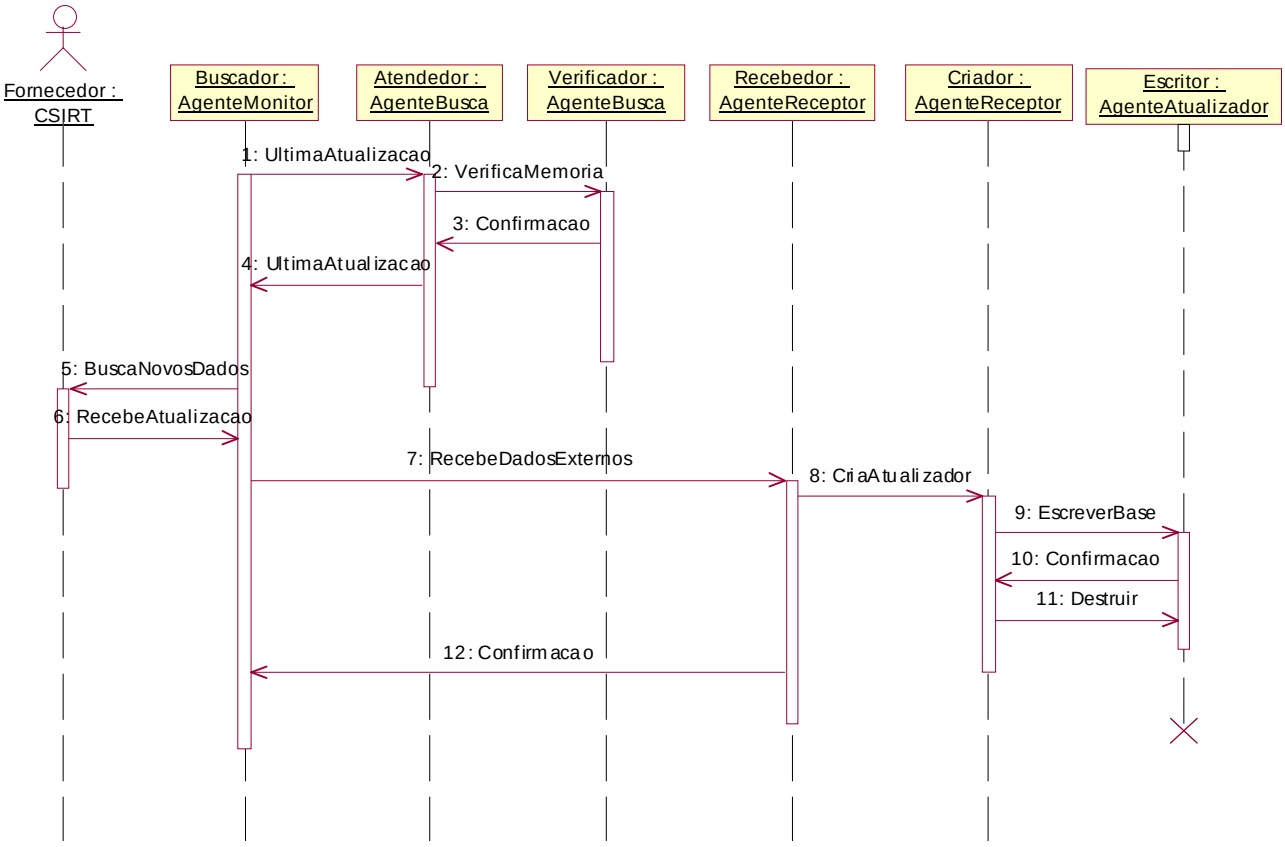

Figura 5.7. Identificação das Responsabilidades dos Agentes a Partir de Diagrama PASSI de Identificação de Responsabilidades. Cenário Obtenção de Dados Externos.

Este cenário, Figura 5.7, dá-se de acordo com os passos descritos a seguir: (1) o *MonitorAgent* busca as últimas atualizações feitas nas bases de dados; o *SeacherAgent* (2,3) busca as últimas atualizações, e (4) responde a solicitação do *MonitorAgent*; de posse do estado atual das bases de dados o

MonitorAgent (5) verifica na entidade externa se existem informações mais recentes, caso positivo, (6) recebe as atualizações a serem feitas e as (7) entrega ao *ReceiverAgent*; o *ReceiverAgent* (8) cria um *UpdateAgent* para proceder o armazenamento; o *UpdateAgent* por sua vez (9) armazena os dados, em seguida o *UpdateAgent* devolve uma (10) confirmação ao *ReceiverAgent* que o criou; o *ReceiverAgent* (11) envia uma mensagem para que o *UpdateAgent* se destrua, e (12) confirma a operação de escrita junto ao *MonitorAgent*.

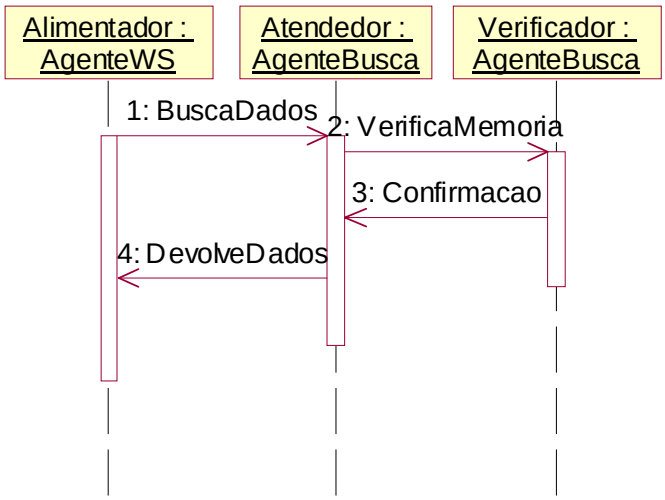

Figura 5.8. Identificação das Responsabilidades dos Agentes a Partir de Diagrama PASSI de Identificação de Responsabilidades. Cenário Compartilhamento de Informações com o Meio Externo.

O segundo cenário, Figura 5.8, alimentação do *Web Services*, os agentes *WSAgent* com o papel de Alimentador e *SearcherAgent* com os papéis de Atendedor e Verificador, trabalham para prover informações a entidades externas ao SDI. A descrição da Figura 5.8, é feita a seguir: O agente *WSAgent* (1) envia uma solicitação ao *SeacherAgent*; O *SeacherAgent* (2) verifica se os dados solicitados estão em memória, e caso (3) positivo, (4) devolve os dados solicitados ao *WSAgent*.

Com o objetivo de reduzir a complexidade dos diagramas mostrados nas Figuras 5.7 e 5.8, em ambos os eventos considerou-se que as informações já estavam carregadas em memória.

5.2.4 Descobrindo as Tarefas de cada Agente

Nesta etapa, deve-se focar no comportamento ordenado de cada agente a fim de decompô-lo em tarefas. Tarefas geralmente encapsulam alguma funcionalidade que forma uma unidade lógica de trabalho. As tarefas que um agente é capaz de executar denotam a capacidade deste agente. A capacidade de cada agente é especificada através de diagramas de atividades, que são Diagramas de Especificação de Tarefas da metodologia PASSI.

Para todos os agentes no modelo, foi desenhado um diagrama de atividade que é composto de duas partes. O lado direito do diagrama contém uma coleção de atividades que simbolizam as tarefas do agente, enquanto que o lado esquerdo contém algumas atividades que representam os outros agentes interagindo com este.

Nos Diagramas de Especificação de Tarefas, Figuras 5.9 e 5.10, resume-se o que o agente é capaz de fazer, ignorando informação sobre quais papéis um agente realiza ao finalizar tarefas em particular.

Para encontrar as tarefas dos agentes observaram-se os Diagramas de Identificação de Papéis ou Responsabilidades, explorando então, todas as interações e ações internas que os agentes executam para realizar um cenário específico. Em cada Diagrama de Identificação de Responsabilidades obtêm-se uma coleção de tarefas relacionadas que são agrupadas de forma apropriada.

A seguir ilustram-se, nas Figuras 5.9 e 5.10, o detalhamento das tarefas dos agentes *SeacherAgent* e *ReceiverAgent*, visto que este são entidades chaves na sociedade que forma o Módulo Gerenciador de Informações.

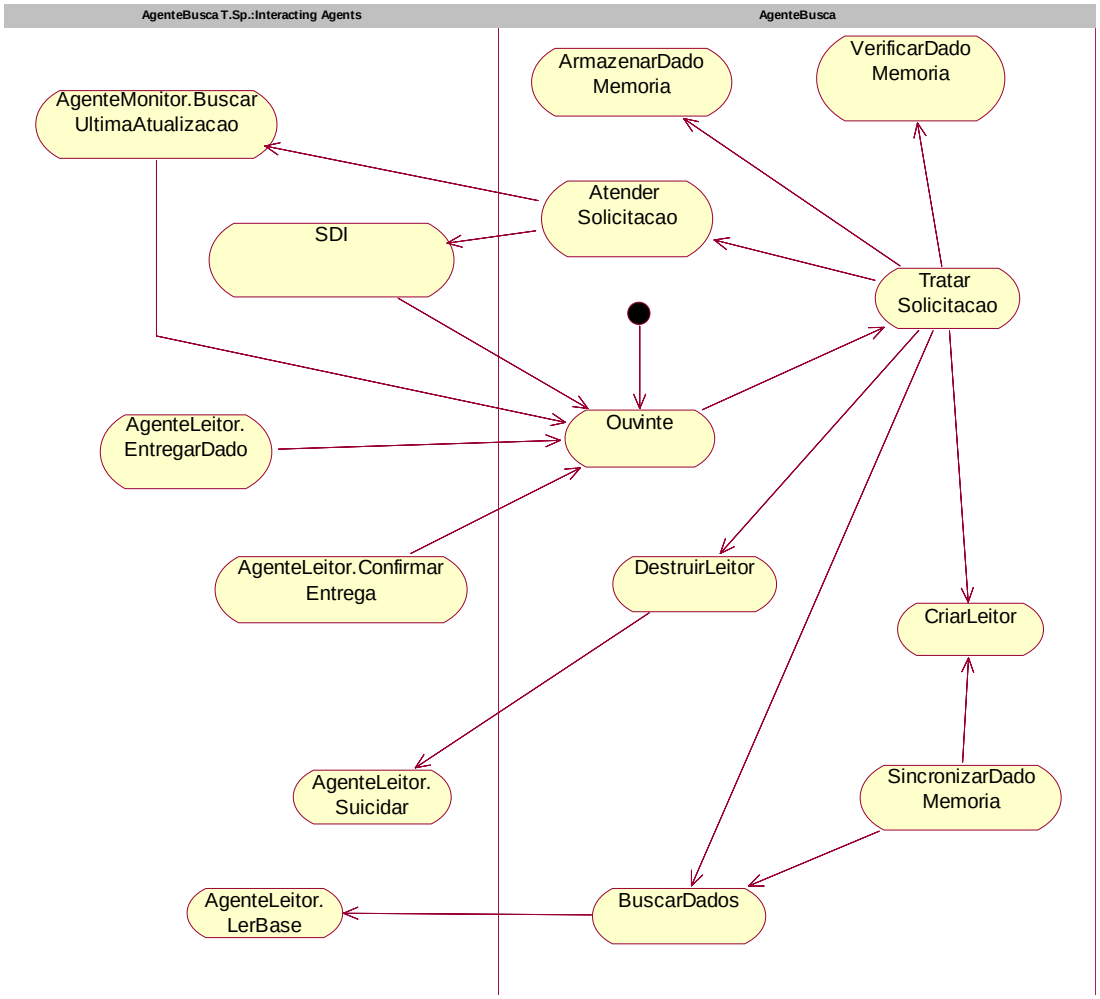

Figura 5.9. Identificação das Tarefas do SeacherAgent Através de Diagrama PASSI de Especificação de Tarefas.

A Figura 5.9 mostra o diagrama de Identificação de Tarefas do *SeacherAgent*. A tarefa *Ouvinte* é necessária para passar as comunicações de entrada à própria tarefa que irá tratá-la, neste caso *TratarSolicitacao.* Foram identificadas as tarefas que se relacionam e controlam os agentes *ReaderAgent* que são *CriarLeitor*, *DestruirLeitor* e *BuscarDados.* Além das

outras tarefas que são realizadas para que o agente desempenhe seu papel dentro da sociedade: *SincronizarDadoMemoria*, *VerificarDadoMemoria*, *ArmazenarDadoMemoria* e *AtenderSolicitacao.*

Além das tarefas do *SeacherAgent*, detalha-se também as tarefas do *ReceiverAgent*. As tarefas identificadas para o *ReceiverAgent*, conforme diagrama da Figura 5.10, foi realizada, assim como o *SearcherAgent* e todos os outro agentes que compõem a sociedade, com base nos Diagramas de Identificação de Responsabilidades.

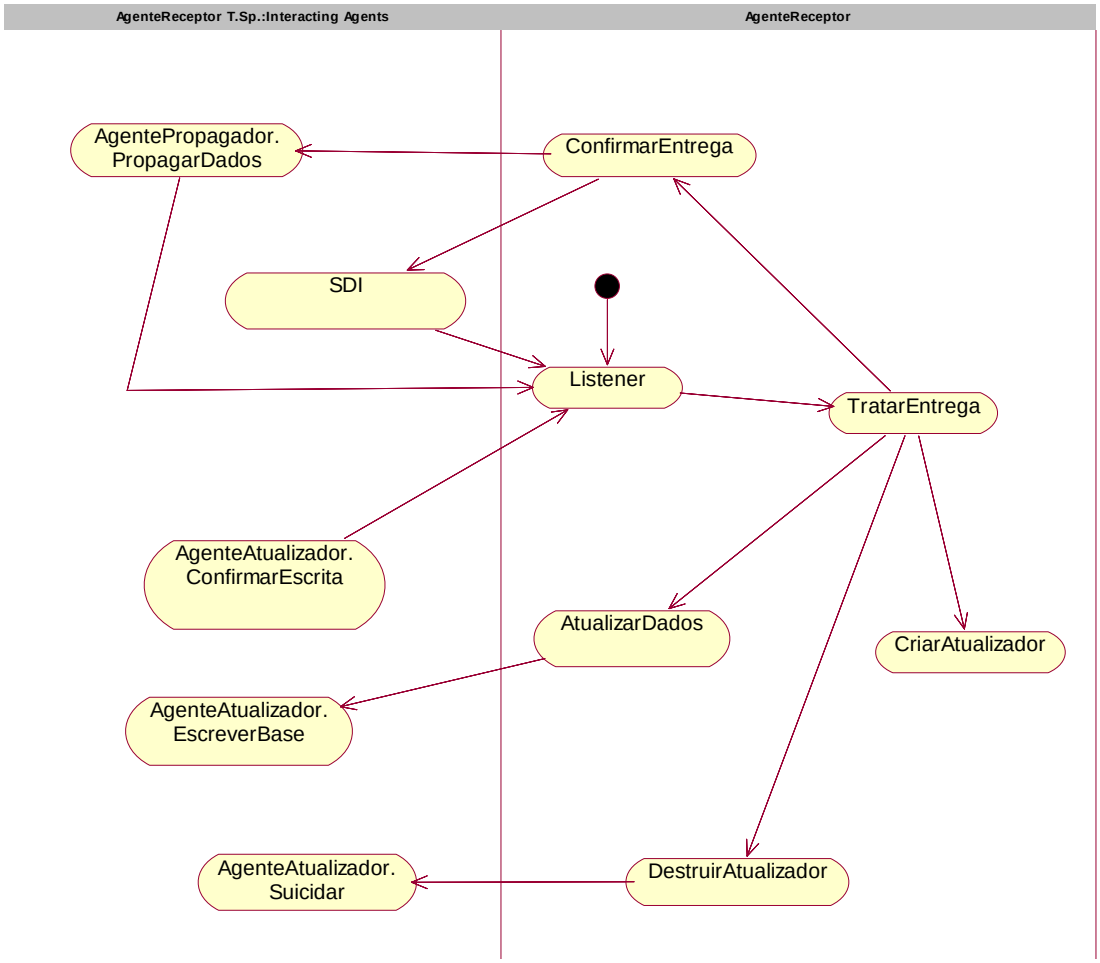

Figura 5.10. Identificação das Tarefas do ReceiverAgent Através de Diagrama PASSI de Especificação de Tarefas.

Na Figura 5.10 apresenta-se o diagrama de Identificação de Tarefas do *ReceiverAgent*. A tarefa *Ouvinte* é usada para passar as comunicações de

entrada à tarefa apropriada para tratá-la, neste caso *TratarEntrega.* Foram identificadas as tarefas que se relacionam e controlam os agentes *UpdateAgent* que são *CriarAtualizador*, *DestruirAtualizador* e *AtualizarDados.* Além destas o ReceiverAgent desempenha ainda a tarefa de *confirmarEntrega.*

5.3 Implementação dos Agentes

Após a fase de análise, definição das funcionalidades, identificação dos agentes, identificação das responsabilidades e detalhamento das tarefas, procedeu-se a implementação do sistema multiagentes.

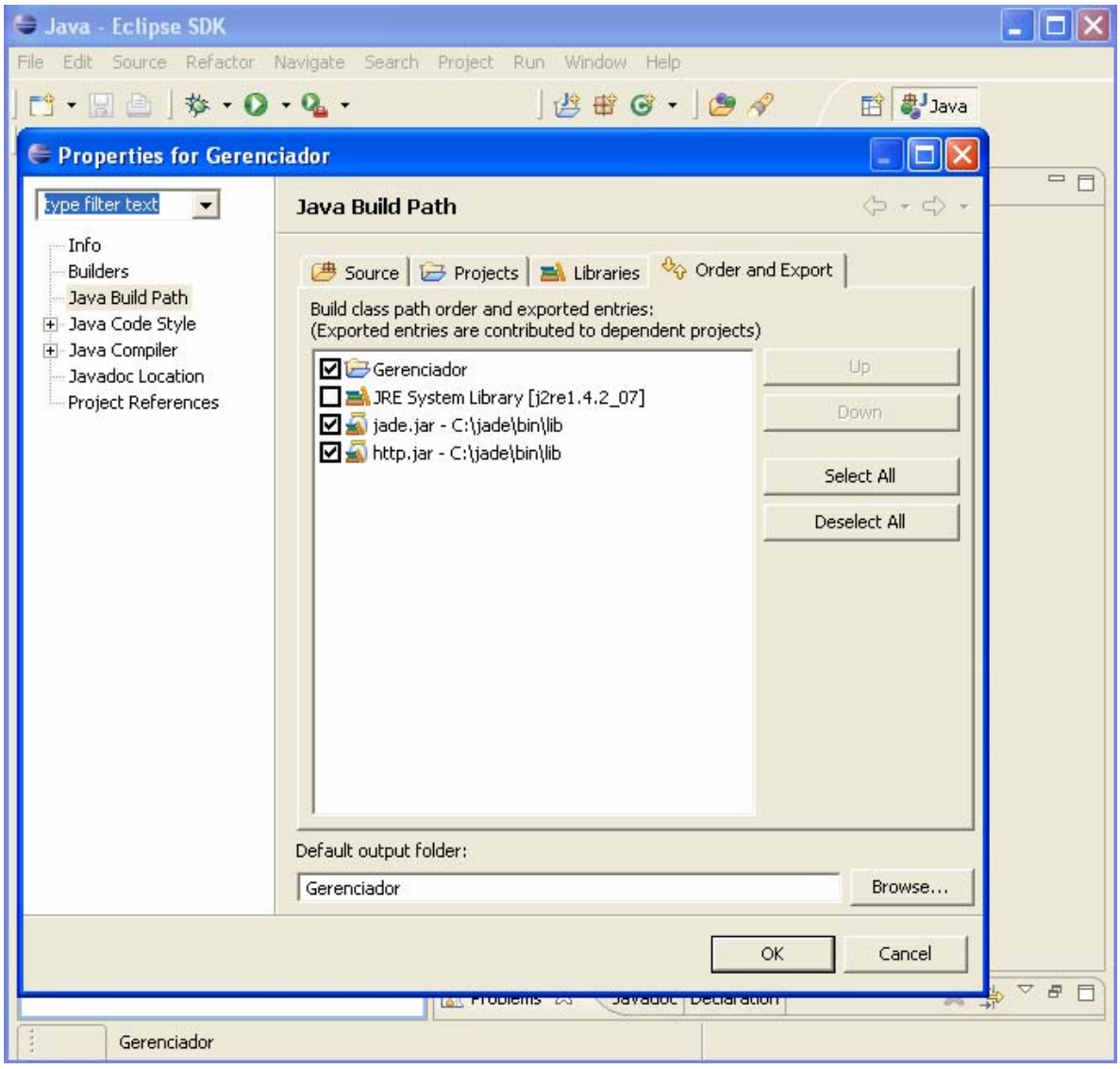

Figura 5.11. Preparação do Eclipse para Utilização do JADE.

Como linguagem de programação utilizou-se o Java [33], por ter dentre outras a característica de ser completamente orientada a objetos e multiplataforma, proporcionando capacidade de programação distribuída para ambientes heterogêneos, característica essencial para domínio em questão.

O JADE, como já mencionado, foi utilizado como a plataforma multiagentes. Esta plataforma traz um conjunto de componentes que dentre outras funcionalidades visa suportar a programação baseada em agentes inteligentes em ambientes distribuídos. Completamente implementada em Java utiliza RMI (*Remote Method Invocation*) como suporte a comunicação remota, além de suportar todas as linguagens e protocolo FIPA para sistemas baseados em agentes.

O ambiente para codificação escolhido foi a IDE (*Integrated Development Environment*) Eclipse [22], um editor para programas Java que aceita o JADE como um *plugin.* O Eclipse é uma IDE gratuita e de código aberto, construída em Java, criada pela IBM em 2001 como um projeto em código aberto. A Figura 5.11 ilustra os pacotes JADE sendo *plugados* ao Eclipse para serem utilizados na construção do Sistema Multiagentes.

Após a escolha e configuração das ferramentas resta definir ainda dois elementos essenciais em um sistema multiagentes: a linguagem de comunicação entre os agentes e o protocolo de interação.

Toda comunicação entre agentes é feita através de troca de mensagens, e uma linguagem para representar essas mensagens deve ser definida. A classe *ACLMessage* da plataforma JADE representa uma mensagem ACL (*Agent Communication Language*) que pode ser trocada entre os agentes. Esta classe contém um conjunto de atributos que são definidos

conforme a especificação FIPA. Como linguagem definida para a comunicação inter-agentes, escolheu-se a LEAP que transmite as mensagens através de um conjunto de *bytes*.

O protocolo de interação, ou seja, o conjunto de regras que devem ser estabelecidas para a comunicação e obedecidas pelo emissor e pelo receptor, foi definido de acordo com o(s) comportamento(s) de cada agente. Assim à medida que os comportamentos dos agentes forem descritos, detalha-se o protocolo definido para a comunicação.

Encontram-se implementados os agentes *SearcherAgent, ReaderAgent, ReceiverAgent, UpdateAgent e TranslatorAgent*. Nas subseções seguintes, os agentes *SeacherAgent* e *ReceiverAgent* terão as suas implementações detalhadas. O objetivo é mostrar como as tarefas identificadas na subseção 5.2.4 foram implementadas. Logo após na subseção 5.3.3 descreve a configuração e o acesso ao gerenciador de banco de dados Xindice.

5.3.1 Implementação do SeacherAgent

A criação do *SeacherAgent*, Figura 5.12, é feita a partir da extensão da classe *Agent* da plataforma JADE.

```
public class SearcherAgent extends Agent{
    /* manipulador de conteudo*/
    private ContentManager manager   = (ContentManager)getContentManager();
    /* linguagem do agente */
    private Codec codec = new LEAPCodec();
    /* ontologia do agente*/
    private Ontology ontology = NidiaOntology.getInstance();
    [...]
```

Figura 5.12. Criação do SeacherAgent a Partir da Plataforma JADE.

Dentre os atributos deste agente, pode-se destacar: *manager* que é o manipulador de conteúdo das mensagens recebidas; *codec*, que recebe uma

instância do *LEAPCodec*, usado para codificar as mensagens; e *ontology* usado para representar uma instância da ontologia definida para a comunicação inter e intra Gerenciador de Informações.

```
protected void setup() {
    try{
        manager.registerLanguage(codec);
        manager.registerOntology(ontology);

        /*adiciona o comportamento que vai atender às solicitações*/
        addBehaviour(new SearcherBehaviour(this));

        /*adiciona o comportamento que vai controlar os ReaderAgents*/
        addBehaviour(new ReaderControlBehaviour(this));

        /*adiciona o comportamento que busca e mantem dados em memoria*/
        addBehaviour(new MemoryRefreshBehaviour(this,60000)); //executado a cada minuto

    }
    catch(Exception e){
        System.out.println("Setup");
        e.getMessage();
    }
}//fim setup
```

Figura 5.13. Método Setup do SeacherAgent.

A Figura 5.13 é usada para ilustrar o método *setup()* do *SeacherAgent*. O método *setup()* é utilizado para conter o código que inicializará o agente.

Este agente possui três comportamentos, conforme se ilustra na Figura 5.14, o *SeacherBeahaviour*, *ReaderControlBehaviour* e o *MemoryRefreshBehaviour.*

```
class SearcherBehaviour extends CyclicBehaviour{

class ReaderControlBehaviour extends CyclicBehaviour{

class MemoryRefreshBehaviour extends TickerBehaviour{
```

Figura 5.14. Comportamentos do SeacherAgent.

O *SearcherBehaviour* é o comportamento principal do *SearcherAgent*. É construído estendendo-se a classe *CyclicBehaviour* que

pertence ao pacote *jade.core.behaviours.* Este comportamento é cíclico, ou seja, é iniciado assim que o agente é iniciado e fica sempre ativo, funcionando como uma espécie de *listener* para o mundo exterior. Ouve os agentes do SDI aguardando por mensagens com performativa do tipo REQUEST, verifica se pode atender a requisição caso as informações já estejam em memória, se não estiver, cria um *ReaderAgent* para atender a solicitação.

As tarefas de *TratarSolicitação*, *AtenderSolicitação*, *VerificarDadoMemoria CriarLeitor* e *BuscarDado* ficam a cargo deste comportamento.

Na Figura 5.15 mostra-se o inicio do comportamento *SeacherBehaviour* que se dá através do método *action()*, padrão na plataforma JADE.

```
/*contem a ação executada pelo comportamento*/
public void action(){
    /* criar modelo de mensagem */
    try{
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage msg = receive(mt);
        if(msg != null){

            try{
                /* verifica conteudo */
                AgentAction action = (AgentAction) manager.extractContent(msg);
                /*cria mensagem de resposta*/
                ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
                reply.setSender(getAID());
                reply.setPerformative(ACLMessage.INFORM);
                reply.setLanguage(codec.getName());
                reply.setOntology(ontology.getName());
          [...]
```

Figura 5.15. Recebimento e Tratamento de Mensagem pelo SeacherBehaviour.

A criação de agentes do tipo *ReaderAgent* é mostrado na Figura 5.16. Através do método *createAgent()* os agentes *SeacherAgent* criam agentes leitores.

```
/*metodo responsavel pela criação do ReaderAgent
 * recebe o identificador do agente que vai ser servido,
 * a ção a ser executada e o identificador do SeacherAgent
 * que o criou
 * */

public void createAgent(AID lord, AgentAction action, AID father){
    try{
        Object[] parametersReaderAgent = new Object[3]; // vetor a ser passado como parametro
        AgentContainer ac = getContainerController(); // container onde o agente será criado
        AgentController agent; // AgentHandler local para o agente a ser criado
        parametersReaderAgent[0] = lord; // endereço do agente a quem vai servir
        parametersReaderAgent[1] = action; //acao a ser realizada
        parametersReaderAgent[2] = father; //endereço do pai
        agent = ac.createNewAgent("reader", "SUA.ReaderAgent", parametersReaderAgent); //nome,
        agent.start(); // cria ReaderAgent
    }
    catch (StaleProxyException e) {
        e.printStackTrace();
    }
}
```

Figura 5.16. Código para Criação de Agentes do Tipo ReaderAgent.

O segundo comportamento, *ReaderControlBehaviour,* se ocupa em executar as tarefas de: ouvir os agentes *ReaderAgent* a fim de controlar e determinar o término de sua execução através da tarefa *DestruirLeitor*; Receber a confirmação da entrega dos dados solicitados por parte do *ReaderAgent*; e enviar mensagem para mesmo suicidar-se. Este comportamento é cíclico, ou seja, é criado junto com o agente e sempre que termina a execução de uma solicitação volta ao estado de pronto.

O terceiro comportamento é *MemoryRefreshBehaviour.* Com o intuito de melhorar o desempenho do serviço foi adicionado ao *SeacherAgent* um comportamento responsável por manter informações em memória.

A partir deste comportamento, podem-se manter em memória de acesso rápido, informações que são ao mesmo tempo pequenas e muito acessadas. Para isso adicionou-se a este agente um comportamento que estende a classe *TickerBehaviour* e executa o método *onTick()* em um dado período de tempo. As informações escolhidas para serem carregadas e mantidas em memória devem ser pré-definidas antes de iniciar a sociedade de

agentes que cooperam e trabalham para compor o Gerenciador de Informações.

```
/* Carregar as bases OKDB e IIDB para a memoria
 * e mantêm-nas atualizadas. As atualizacoes acontecem
 * a cada minuto
 * */
class MemoryRefreshBehaviour extends TickerBehaviour{

    public MemoryRefreshBehaviour(Agent a, long b){
        super(a,b);
    }

    protected void onTick(){
     [...]
```

Figura 5.17. Codificação Comportamento MemoryRefreshBehaviour.

A Figura 5.17 mostra o comportamento *MemoryRefreshBehaviourI*. As tarefas de *ArmazenarDadoMemoria* e *SincronizarDadoMemoria* são executadas por este comportamento.

5.3.2 Implementação do ReceiverAgent

A implementação do *ReceiverAgent*, assim como a dos demais agentes que compõem a sociedade de agentes que formam o Gerenciador de Informações, segue os mesmos moldes do *SeacherAgent*, ficando a principal distinção a cargo do(s) comportamento(s) que cada agente executa.

```
public class ReceiverAgent extends Agent{
    /* manipulador de conteudo*/
    private ContentManager manager   = (ContentManager)getContentManager();
    /* linguagem do agente */
    private Codec codec = new LEAPCodec();
    /* ontologia do agente*/
    private Ontology ontology = NidiaOntology.getInstance();
        [...]
```

Figura 5.18. Código para Criação do ReceiverAgent a Partir da Plataforma JADE.

A Figura 5.18 é usada para mostrar o código utilizado para criação do *ReceiverAgent*. Como qualquer outro agente sobre a plataforma JADE, é criado estendendo a classe *Agent* do pacote *jade.core,* os atributos na figura possuem a mesma semântica dos atributos do *SeacherAgent*.

O ponto de início na execução do agente é o método *setup()*, que é ilustrado a seguir na Figura 5.19. Este agente possui três comportamentos, *ReceiverBehaviour*, *UpdateControlBehaviour* e *ExternalDataControl*. Estes comportamentos são responsáveis por executar as tarefas detalhadas na Figura 5.10.

```
protected void setup() {
    try{
        manager.registerLanguage(codec);
        manager.registerOntology(ontology);

        /*adiciona o comportamento que vai atender ao SDI*/
        addBehaviour(new ReceiverBehaviour(this));

        /*adiciona o comportamento que vai controlar agentes UpdateAgent*/
        addBehaviour(new UpdateControlBehaviour(this));

        /*adiciona o comportamento que vai receber dados replicados e dados externos*/
        addBehaviour(new ExternalDataControl(this));

    }
    catch(Exception e){
        System.out.println("Setup");
        e.getMessage();
    }
}//fim setup
```

Figura 5.19. Método Setup do ReceiverAgent.

O *ReceiverBehaviour*, Figura 5.20, estende a classe *CyclicBehaviour* e implementa um comportamento cíclico. As tarefas *TratarEntrega, CriarAtualizador, AtualizarDados* e *ConfirmarEntrega* são executadas por este comportamento. Assim, o comportamento *ReceiverBehaviour* é responsável por receber as solicitações dos agentes do SDI e atendê-las. Aguarda por mensagens com performativa REQUEST, e procede a criação de agentes do tipo *UpdateAgent* para execução das ações requisitadas.

Já o comportamento *ExternalDataControl* é responsável pelas mesmas tarefas: *TratarEntrega, CriarAtualizador, AtualizarDados* e *ConfirmarEntrega.* Porém com mensagens oriundas do *MonitorAgent* e de dados do *PropagatorAgent*. O motivo de existir dois comportamentos que

realizam as mesmas tarefas deve-se ao fato destas fontes já entregarem dados devidamente tratados. O *ExternalDataControl* também é um comportamento cíclico.

```
/* Comportamento responsavel por receber as solicitacoes dos agentes
 * do SDI e atende-las. aguarda por mensagens com performativa REQUEST,
 * cria um UpdateAgent que executará a ação requisitada
 *
 */
class ReceiverBehaviour extends CyclicBehaviour{

    public ReceiverBehaviour(Agent a){
        super(a);
    }

    public void action(){
        /* criar modelo de mensagem */
        try{
            MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
            ACLMessage msg = receive(mt);

            if(msg != null){
                /*verifica conteudo*/
                AgentAction action = (AgentAction) manager.extractContent(msg);
                if(action instanceof DataBaseUpdate){
                    /*cria agente escravo e passa a ação a ser executada*/
                    createAgent(msg.getSender(),action,getAID());
                }
            }
            else{ /* caso não haja mensagens o comportamente sede os recursos para
                   * os outros comportamentos do Agente
                   */
                block();
            }

        }
            [...]
```

Figura 5.20. Comportamento ReceiverBehaviour do ReceiverAgent.

O terceiro comportamento, o *UpdateControlBehaviour*, do ReceiverAgent, assim como os demais é um comportamento cíclico, que em cada mensagem recebe a confirmação da execução da tarefa repassada, avalia se há trabalho pendente, caso positivo envia para o *UpdateAgent* realizar novas atualizações nas bases de dados, caso negativo envia mensagem ordenando a auto destruição do *UpdateAgent*. As tarefas de *DestruirAtualizador* fica a cargo deste comportamento, assim como o monitoramento e controle dos agentes *UpdateAgent.*

Na Figura 5.21 mostra-se o código utilizado pelo agente *ReceiverAgent* para criar sob demanda agentes do tipo *UpdateAgent*.

```
/* Metodo responsavel por criar agentes do tipo UpdateAgent */
public void createAgent(AID lord, AgentAction action, AID father){
    try{
        Object[] parametersReaderAgent = new Object[3]; // vetor a ser passado como parametr
        AgentContainer ac = getContainerController(); // container onde o agente será criado
        AgentController agent; // AgentHandler local para o agente a ser criado
        parametersReaderAgent[0] = lord; // endereço do agente a quem vai servir
        parametersReaderAgent[1] = action; //acao a ser realizada
        parametersReaderAgent[2] = father; //endereço do pai
        agent = ac.createNewAgent("update", "SUA.UpdateAgent", parametersReaderAgent); //nome
        agent.start(); // cria ReaderAgent
    }
    catch (StaleProxyException e) {
        e.printStackTrace();
    }
} // fim createAgent
```

Figura 5.21. Código para Criação de UpdateAgent na plataforma JADE.

5.3.3 Configuração das bases de dados utilizando Xindice

O Xindice é a continuação do projeto inicialmente chamado de *dbXML Core*. O código fonte do *dbXML Core* foi doado à Fundação de Software Apache em dezembro de 2001 [62].

O gerenciador de dados Xindice trabalha com o conceito de Coleções de Documentos. Estas Coleções são organizadas hierarquicamente, semelhantes ao sistema de arquivos de um sistema operacional, onde uma Coleção é um objeto semelhante a um diretório, e um Documento é um objeto semelhante a um arquivo. Cada Coleção pode conter um ou mais Documentos e também outras Coleções.

O Xindice é um servidor de banco de dados projetado para armazenar dados em formato XML, que utiliza *XPath* [65] como linguagem de consulta, e traz uma API que suporta Documentos DOM (*Document Object Model*) e Eventos SAX (*Simple API for XML*). Para auxiliar a administração dos dados provê também uma ferramenta de linha de comando.

Através de XPath pode-se realizar buscas a nível de Coleção ou a nível de Documento. Isso permite que vários Documentos sejam lidos em busca de uma determinada informação.

A partir da versão 1.1 Xindice não é mais um servidor *standalone*. As suas funções passaram a ser providas através de um servidor de aplicação. Xindice é recomendado para ser usado com os servidores de aplicação Tomcat e Jetty. Assim, neste trabalho utilizou-se Xindice a partir do servidor de aplicação Tomcat com JDK 1.4.

5.3.3.1 Criação das Coleções para armazenamento dos dados

Após a instalação e configuração do Xindice, procedeu-se a criação das Coleções que representam cada base de dados do domínio em questão, conforme padronização proposta no Capítulo 3.

Na Figura 5.22, apresentam-se os comandos usados diretamente na ferramenta de linha de comandos do Xindice para criar quatro Coleções: *iidb,* usada para armazenar as informações de Assinaturas de Ataque; *dfdb,* usada para armazenar as informações de Dados Forenses; *radb,* usada para armazenar informações de Ações de Resposta; e *stdb,* que é a Coleção destinada a armazenar dados de Estratégias de Ações de Resposta.

```
xindice add_collection -c /db -n iidb
xindice add_collection -c /db -n dfdb
xindice add_collection -c /db -n radb
xindice add_collection -c /db -n stdb
```

Figura 5.22. Código para Criação das Coleções de Dados.

5.3.3.2 Acessando o Xindice

Assim como em uma base de dados relacional, para acessar informações no Xindice através de uma aplicação, faz-se necessário utilizar um

driver fornecido junto com o Xindice para estabelecer comunicação com a base de dados.

O código usado pelo *ReaderAgent* para carregar o *driver*, criar um objeto Database e registrar o objeto Database criado é ilustrado na Figura 5.23. A criação dos objetos para o estabelecimento de conexão com banco de dados é feito a partir da importação do pacote *org.xmldb.api*.

```
    [...]
/* objetos para acesso ao Xindice */
private Collection col = null;
private String driver = null;
private Class classe = null;
private Database database = null;
private String uri = null;
private String xPath = null;
private XPathQueryService service = null;
private ResourceSet resultSet = null;
private ResourceIterator results = null;
   [...]
protected void setup() {
    try{
        Object[] args = getArguments();
        if(args.length == 3){
            lordAgent = (AID) args[0];
            System.out.println("READER - lordAgent " +lordAgent.getName());
            action  = (AgentAction) args[1];
            System.out.println("READER - action");
            fatherAgent = (AID) args[2];
            System.out.println("READER - fatherAgent " +fatherAgent.getName());
        }
        else{
            System.out.println("READER - Numero de parametros iniciais != 3");
            System.out.println("READER - encerrando");
            doDelete(); //encerra o agente
        }

        driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
        classe = Class.forName(driver);
        database = (Database) classe.newInstance();
        DatabaseManager.registerDatabase(database);
    [...]
```

Figura 5.23. Código Utilizado Para Estabelecimento de Conexão com um Banco de Dados Xindice.

A parte superior da Figura 5.23 mostra a declaração dos atributos necessários para o estabelecimento da conexão do *ReaderAgent* com a base de dados. Estes atributos são: *col,* é um atributo do tipo *Collection* usado para representar uma Coleção do banco de dados; *driver*, é um atributo do tipo

String usado para armazenar a localização do *driver*; *classe,* é um atributo usado para "carregar" o *driver*; *database,* é um atributo do tipo *Database* e é usado para instanciar a base de dados; *uri,* é um atributo do tipo *String* usado para armazenar a Coleção que deve ser lida ou escrita; *xPath,* é um atributo usado para armazenar a instrução a ser processada; *service,* representa o serviço *XPathQueryService* que é fornecido pelo Xindice para execução de comandos do tipo *XPath*; *resultSet;* e *results,* são atributos de tipo *ResourceSet* e *ResourceIterator* respectivamente e são usados para receber e manipular os dados extraídos do banco de dados.

Apresenta-se ainda na Figura 5.23, o método *setup()* do *ReaderAgent* onde encontra-se o código usado para carregamento do *driver* e registrar o tipo *Database.*

A Figura 5.24 ilustra o armazenamento de um Documento DOM. Este código é utilizado pelo *UpdateAgent* para escrever na base de dados um arquivo em formato XML, que em memória, é representado como um Documento DOM.

```
    [...]
/* conexão com o banco para armazenar documento */
col = DatabaseManager.getCollection(uri);
XMLResource document = (XMLResource) col.createResource(null, "XMLResource");
document.setContentAsDOM(doc);
col.storeResource(document);
   [...]
```

Figura 5.24. Código Utilizado Para Escrever um Documento em Meio Persistente.

O objeto *doc* é do tipo *Document* e contém um Documento DOM a ser inserido na base de dados, conforme mostra a Figura 5.24. Este Documento é gerado pelo *TranslatorAgent* quando recebe uma solicitação de converter uma informação de formato ACL para formato XML. Apresenta-se na

Figura 5.25 parte do código usado pelo *TranslatorAgent* para realizar esta conversão de dados.

```
      [...]
/* metodo responsavel por gerar XML em formato IIDB */
private void MakeXML(Attack attack, Document doc, Element event){
    try{
        // cria nó identifier
        Node identifier = doc.createElement("identifier");
        identifier.appendChild(doc.createTextNode(attack.getIdentifier()));
        event.appendChild(identifier);
        // cria nó name
        Node name = doc.createElement("name");
        name.appendChild(doc.createTextNode(attack.getName()));
        event.appendChild(name);
        // cria nó description
        Node description = doc.createElement("description");
        description.appendChild(doc.createTextNode(attack.getDescription()));
        event.appendChild(description);
        // cria nó impact
        Node impact = doc.createElement("impact");
        impact.appendChild(doc.createTextNode(attack.getImpact()));
        event.appendChild(impact);
        // cria nó severity
        Node severity = doc.createElement("severity");
        severity.appendChild(doc.createTextNode(attack.getSeverity()));
        event.appendChild(severity);
        // cria nó cve
        Node cve = doc.createElement("cve");
        cve.appendChild(doc.createTextNode(attack.getCve()));
        event.appendChild(cve);

        // cria elemento tool e subelementos
        Element tool = (Element)doc.createElement("tool");
        tool.setAttribute("name", attack.getTool().getName());
        tool.setAttribute("description", attack.getTool().getDescription());
        event.appendChild(tool);
      [...]
```

Figura 5.25. Método MakeXML() do TranslatorAgent para Transformação de Informações para o Formato XML.

O método *MakeXML()* é um método sobrecarregado que é usado pelo *TranslatorAgent* para converter dados que estão em formato ACL para formato XML. Na Figura 5.25 ilustra-se o método *MakeXML(),* sendo utilizado para gerar um documento XML para a base de dados de Assinaturas de Ataque. Após a execução deste método, o *TranslatorAgent* devolve um objeto do tipo *Document* ao agente que solicitou a conversão de formato.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.incident.com/cap/1.0"
    xmlns:cap="http://www.incident.com/cap/1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <element name="attack">
    - <annotation>
        <documentation>Mensagem de atualizacao da IIDB (versao 1.0)</documentation>
      </annotation>
    - <complexType>
        <element name="idenfier" type="string" />
        <element name="name" type="string" />
        <element name="description" type="string" />
        <element name="impact" type="string" />
      - <element name="severity">
        - <simpleType>
          - <restriction base="string">
              <enumeration value="Extreme" />
              <enumeration value="Severe" />
              <enumeration value="Moderate" />
              <enumeration value="Minor" />
              <enumeration value="Unknown" />
            </restriction>
          </simpleType>
        </element>
        <element name="cve" type="string" minOccurs="0" />
      - <element name="tool" minOccurs="0" maxOccurs="unbounded">
        - <complexType>
          - <sequence>
              <element name="name" type="string" minOccurs="0" />
              <element name="description" type="string" minOccurs="0" />
            </sequence>
          </complexType>
        </element>
                    [...]
```

Figura 5.26(a). Esquema XML Para Informações de Assinaturas de Ataques.

A codificação apresentada na Figura 5.25 segue o formato definido do Esquema XML, conforme mostra as Figuras 5.26(a) e 5.26(b), para dados da base de dados de Assinaturas de Ataques.

Após apresenta-se na Figura 5.24 a realização de uma operação de inserção de um Documento em um banco de dados XML gerenciado pelo Xindice; falta ainda mostrar como realizar a busca por um determinado Documento. A Figura 5.27 traz o código usado pelo *ReaderAgent* para buscar um determinado Documento em uma Coleção específica.

```
            [...]
    - <element name="expression" minOccurs="0" maxOccurs="unbounded">
      - <complexType>
        - <sequence>
            <element name="description" type="string" minOccurs="0" />
            <element name="amount" type="int" minOccurs="0" />
          </sequence>
        </complexType>
      </element>
    - <element name="process" minOccurs="0" maxOccurs="unbounded">
      - <complexType>
        - <sequence>
            <element name="pid" type="string" minOccurs="0" />
            <element name="name" type="string" minOccurs="0" />
            <element name="platform" type="string" minOccurs="0" />
          </sequence>
        </complexType>
      </element>
    - <element name="service" minOccurs="0" maxOccurs="unbounded">
      - <complexType>
        - <sequence>
            <element name="name" type="string" minOccurs="0" />
            <element name="port" type="int" minOccurs="0" />
            <element name="protocol" type="string" minOccurs="0" />
            <element name="platform" type="string" minOccurs="0" />
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>
</schema>
```

Figura 5.26(b). Esquema XML Para Informações de Assinaturas de Ataques.

O *ReaderAgent* para realizar a busca de um Documento que está em disco, executa os seguintes passos: escreve no atributo *uri* o endereço da Coleção que contém o Documento a ser buscado; modifica o atributo *col* para "apontar" para o endereço que estar em *uri*; faz *xPath* receber a instrução a ser processada, que no exemplo apresentado na Figura 5.27 é "//iidb[identifier='111']" e, significa que qualquer Documento dentro da Coleção *iidb* que possua um elemento chamado *identifier* com valor igual a "111" será selecionado, e entregue ao agente; executa o método *query()* de um objeto *XPathQueryService* para processar a instrução que estar em *xPath*; e usa um atributo do tipo *ResourceSet* para receber o resultado da busca.

```
        [...]
/* conexão com a coleção iidb e busca de documento */
try{
    uri = "xmldb:xindice:///db/iidb";
    col = DatabaseManager.getCollection(uri);
    xPath = "//iidb[identifier='111']";
    service = (XPathQueryService) col.getService("XPathQueryService", "1.0");
    resultSet = service.query(xPath);
    results = resultSet.getIterator();
        [...]
```

Figura 5.27. Código Utilizado Para Buscar um Documento em Meio Persistente.

Esta busca retorna um Documento XML semelhante ao apresentado na Figura 5.28.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <attack>
    <identifier>111</identifier>
    <name>ataque01</name>
    <description>intrusão causada por trojan</description>
    <impact>inconsitencia no sistema de arquivos</impact>
    <severity>severe</severity>
  - <expression>
      <description>remove all</description>
      <amount>3</amount>
    </expression>
  - <process>
      <name>Trojan-PSW.Win32.Agent.im</name>
      <platform>windows</platform>
    </process>
  - <service>
      <name>WQEKEZF</name>
      <port>1521</port>
      <protocol>TCP</protocol>
      <platform>windows</platform>
    </service>
  </attack>
```

Figura 5.28. Exemplo de Documento XML da Base de Dados de Assinaturas de Ataques.

5.4 Conclusões

Neste Capítulo mostrou-se o processo de implementação do Módulo Gerenciador de Informações. Para tanto, utilizou-se a metodologia de desenvolvimento de sistemas multiagentes PASSI, a linguagem de

programação Java através do framework de desenvolvimento de agentes JADE e o gerenciador de banco de dados XML Xindice.

A metodologia PASSI, mostrou-se eficiente no processo de concepção do Sistema Multiagentes. Através de denotação UML, foi possível a identificação dos agentes e o detalhamento das responsabilidades e tarefas de cada agente. Com PASSI, foi também concebida uma ontologia usada pelo Sistema Multiagentes.

Na implementação, o JADE se mostrou uma plataforma de fácil utilização, pois traz um conjunto de componentes que fornece um bom nível de abstração ao programador, tendo este que se preocupar somente com a implementação no que se refere às tarefas e comportamentos dos agentes.

Assim, detalhou-se a implementação dos agentes *SeacherAgent* e *ReceiverAgent*, para demonstrar o processo de implementação. Além desses, os agentes *ReaderAgent, UpdateAgent* e *TranslatorAgent* também tiveram seus códigos escritos.

Mostrou-se ainda, a configuração do gerenciador de banco de dados XML Xindice e como os agentes *ReaderAgent* e *UpdateAgent* realizam a escrita e leitura de Documentos armazenados.

6 INTEGRAÇÃO E TESTES

Neste Capítulo, descreve-se a integração do Gerenciador de Informações ao NIDIA (*Network Intrusion Detection System Based on Intelligent Agent*). Como descrito no Capítulo 2, o NIDIA é um SDI (Sistema de Detecção de Intrusão) baseado em agentes inteligentes e estruturado em camadas com responsabilidades bem definidas. Com a integração do Módulo Gerenciador de Informações ao NIDIA as camadas de Atualização e Armazenamento ficam sob a responsabilidade do novo módulo.

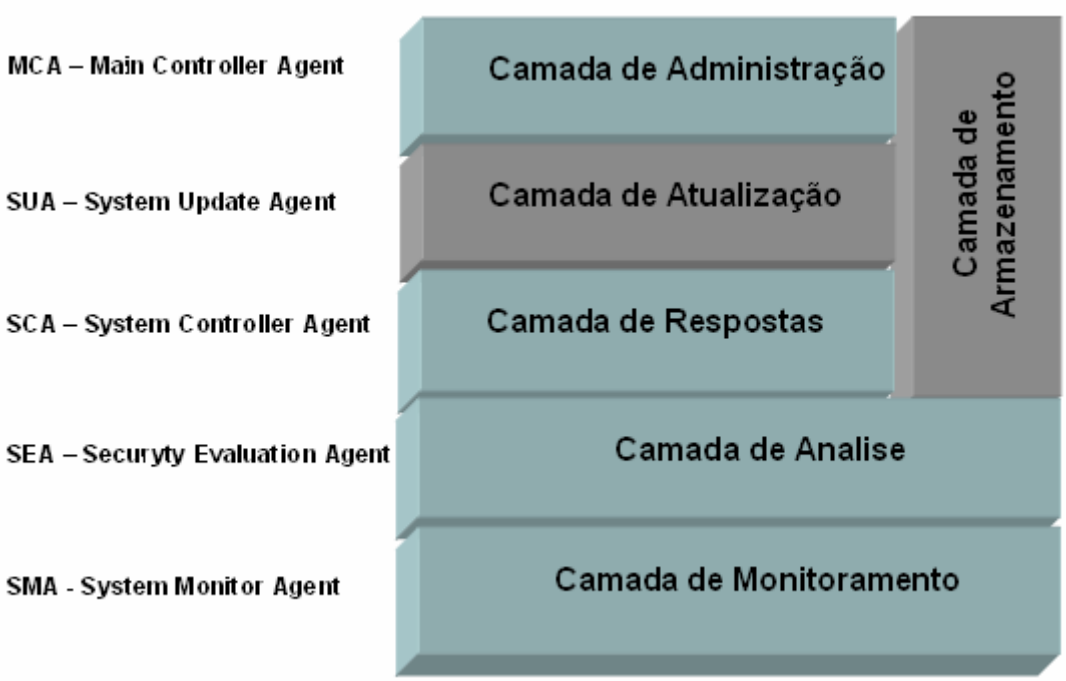

Figura 6.1. Camadas do NIDIA que Ficam Sob a Responsabilidade do Módulo Gerenciador de Informações.

Usou-se o Módulo Gerenciador de Informações [15,16] para prover as funcionalidades inerentes à camada de Armazenamento, ou seja, manter de forma persistente informações provenientes das demais camadas, garantindo segurança e confiabilidade aos dados.

Quanto à camada de Atualização o foco foi o gerenciamento de informações geradas internamente. Todas as funcionalidades que devem ser

executadas por agentes NIDIA do tipo SUA (*System Update Agent*), serão realizadas por agentes do Módulo Gerenciador de Informações.

Com a integração, as bases de dados definidas no Capítulo 3, receberam siglas, que são:

- Base de Intrusões passa a chama-se IIDB ou *Intrusion Information Data Base*;
- Base de Ações de Resposta recebeu a sigla RADB que significa *Response Action Data Base*;
- Base de Estratégia passa a denominar-se STDB ou *Strategy Data Base*;
- Base de Dados Forenses, na integração com o NIDIA recebeu a denominação *Data Forense Data Base* ou DFDB;
- OKDB, do inglês *Optimized KeyWord Data Base*, é a Base de Palavras Chave. O NIDIA é um SDI com mecanismo de detecção por assinatura, a partir de informações coletadas em *logs* ou pacotes de rede, assim, utiliza-se desta base para acelerar o processo de identificação de intrusões.

Deste ponto em diante, a referência às bases de dados de um SDI far-se-á através das siglas descritas acima.

6.1 Construção de Pseudo-NIDIA

O NIDIA é um projeto em andamento, e dentre as áreas atualmente pesquisadas e alvos de trabalhos em nível de mestrado pode-se citar:

- Mecanismos de Tolerância a Falhas [55];
- Transmissão de Mensagens Seguras [43];

- Introdução de Gerenciamento Remoto através de Web Services e MDA (*Model Driven Architecture*) [54].

Assim, alguns dos componentes estão sendo construídos em outros trabalhos, ou ainda em nível de pesquisa.

Como o objetivo é testar o Gerenciador proposto [15,16] e garantir que ele interagirá com SDIs multiagentes como o NIDIA, realizando as tarefas a que está proposto, desenvolveu-se um protótipo do NIDIA. Um agente, denominado *NidiaAgent*, que simula o comportamento de cada uma das camadas do NIDIA ao interagir com o Módulo Gerenciador de Informações.

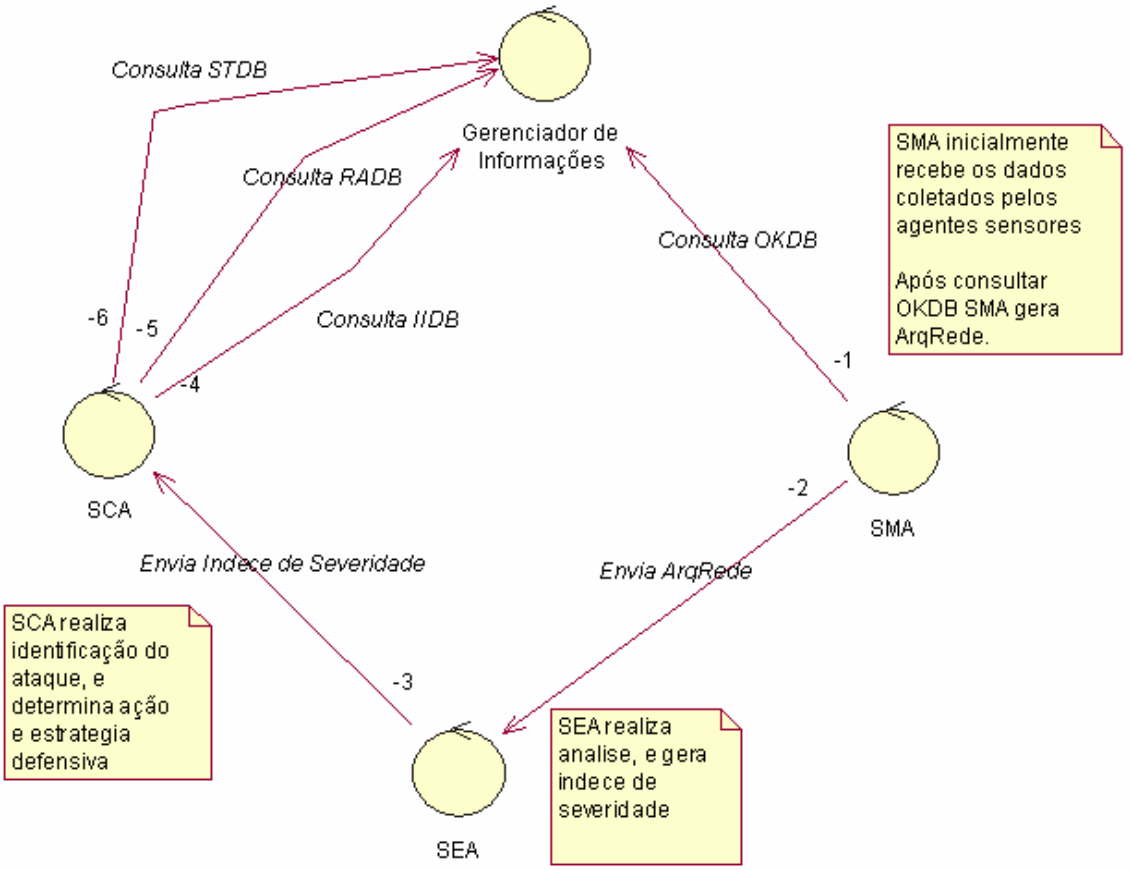

Figura 6.2. Comportamento do NIDIA Simulado pelo NidiaAgent.

O comportamento do *NidiaAgent* é descrito na Figura 6.2. As ações simuladas pelo *NidiaAgent,* correspondem as ações do SMA (*System Monitoring Agent*) e do SCA (*System Controller Agent*) que se utilizam das

bases de dados, através do Gerenciador de Informações, para identificação de intrusão e determinação de ação defensiva.

6.1.1 Implementação do NidiaAgent

O *NidiaAgent* foi construído com um único comportamento. Este comportamento estende a classe *Behaviour*, e implementa um comportamento genérico. Um comportamento genérico é controlado por estados e executa operações diferentes que dependem daquele estado. Eles se completam quando uma determinada condição é conhecida. Assim, dependendo do estado, o *NidiaAgent* reage e age de forma diferente.

```
public class NidiaAgent extends Agent{
    /* manipulador de conteudo*/
    private ContentManager manager  = (ContentManager)getContentManager();
    /* linguagem do agente */
    private Codec codec = new LEAPCodec();
    /* ontologia do agente*/
    private Ontology ontology = NidiaOntology.getInstance();
    /* objetos usados na geracao das informacoes forenses */
    private Info info = null;
    private ForenseData forenseData = null;
    /* agente que está atendendo às solicitações*/
    private AID slaveAgent = null;
    private AID searcher = null;
    /* usados para capturar e formatar datas */
    private GregorianCalendar gc = null;
    private SimpleDateFormat dateFormat = null;

    class RequestPerformerBehaviour extends Behaviour{
        /* marca os passos para identificação e resolução do ataque */
        private int step = 0;
        /* modelo de mensagem a ser recebida */
        private MessageTemplate mt;
              [...]
```

Figura 6.3. Código de Criação do NidiaAgent e do Comportamento RequestPerformerBehaviour a Partir da Plataforma JADE.

A Figura 6.3 é utilizada para ilustrar a criação do agente *NidiaAgent* a partir da extensão da classe *Agent.* A Figura 6.3 também ilustra a criação do comportamento *RequestPerformerBehaviour* a partir da extensão da classe *Behaviour.*

```
protected void setup() {
    manager.registerLanguage(codec);
    manager.registerOntology(ontology);

    /*adiciona o comportamento que vai interagir com o
     * Gerenciador de Informações
     */
    addBehaviour(new RequestPerformerBehaviour(this));

}//fim setup
```

Figura 6.4. Método Setup() do NidiaAgent.

Mostra-se na Figura 6.4, o método *setup()* do *NidiaAgent*. Este método é o ponto de início da execução de qualquer agente na plataforma JADE. No método *setup()* também é declarado o comportamento *RequestPerformerBehaviour*.

```
                [...]
switch(step){
    case 0:
        try{
            /* busca a base de palavras chave */
            request.addReceiver(searcher); // primeira contato com o SeacherAgent

            DataBaseSearcher dataBaseName = new DataBaseSearcher();
            dataBaseName.setDataBaseName("OKDB");
            System.out.println("NIDIA - Buscar Base de Palavras Chave");
            manager.fillContent(request,dataBaseName);
            send(request);

            step = 1;
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        break;
    case 1:
        /* recebe base de palavras chave,
         * gera arquivo de rede e  busca ataque
         */
        response = receive(mt);
        if(response != null){
            try{
                slaveAgent = response.getSender(); // SearcherAgent ou ReaderAgent
                DataBase wordBase = null;
                Object csWordBase = response.getContentObject();
                if (csWordBase instanceof DataBase){
                    wordBase = (DataBase) csWordBase;
                    List listWord = new ArrayList();
                    listWord = wordBase.getBase();
                 [...]
```

Figura 6.5. Parte do Código do Comportamento RequestPerformerBehaviour do Agente NidiaAgent.

Parte da implementação do comportamento *RequestPerformerBehaviour* pertencente ao *NidiaAgent* ilustra-se na Figura 6.5. Uma estrutura de múltiplas escolhas, *switch-case*, é utilizada para controlar os vários estados que este comportamento contém.

Por não ser um comportamento cíclico, ou seja, executa uma única vez e por *default* finaliza, o *RequestPerformerBehaviour* necessita ter a sua finalização devidamente tratada. Isto é feito através da introdução dos métodos *done()* e *end()* ao comportamento. Através destes métodos, pode-se tratar o fim da execução de um determinado comportamento.

O código escrito para implementar os métodos *done()* e *end()* é mostrado na Figura 6.6.

```
public boolean done() {
    if(step == 5 || step == 9){
        End();
        return true;
    }
    return false;

    //return (step == 3 || step == 9);
}//fim done

/* metodo executado no fim da acao do comportamento*/
public void End(){
    /* Verifica se o agente da comunicacao é o Searcher,
     * caso contrario - Reader, envia uma mensagem de CANCEL
     */
    if (!slaveAgent.equals(searcher)){
        try{
            ACLMessage end = new ACLMessage(ACLMessage.CANCEL);
            end.setSender(getAID());
            end.setPerformative(ACLMessage.CANCEL);
            end.setLanguage(codec.getName());
            end.setOntology(ontology.getName());
            end.addReceiver(slaveAgent);
            end.setContent("Finalizar");
            System.out.println("NIDIA - Enviando mensagem para slave encerrar");
            send(end);
        }
        catch(Exception e){
            System.out.println("onEnd "+e.getStackTrace()+" "+e.getMessage());
        }
    }
    doDelete();
}
```

Figura 6.6. Implementação dos Métodos done() e end() do Comportamento RequestPerformerBehaviour do Agente NidiaAgent.

6.2 Testes

Na realização dos testes, utiliza-se uma base não replicada previamente alimentada com informações de Assinaturas de Ataque, Ações de Respostas e Estratégias para execução das ações.

Com o intuito de alcançar um único significado semântico ao vocabulário utilizado pelos agentes, foi ainda desenvolvida uma ontologia baseada nos principais conceitos do domínio, conforme subseção 5.2.2 do Capítulo 5. A partir da ontologia foi possível especificar a linguagem e o protocolo de interação. O pacote *jade.content.onto.basic* inclui um conjunto de classes que foram utilizadas como suporte a implementação da ontologia.

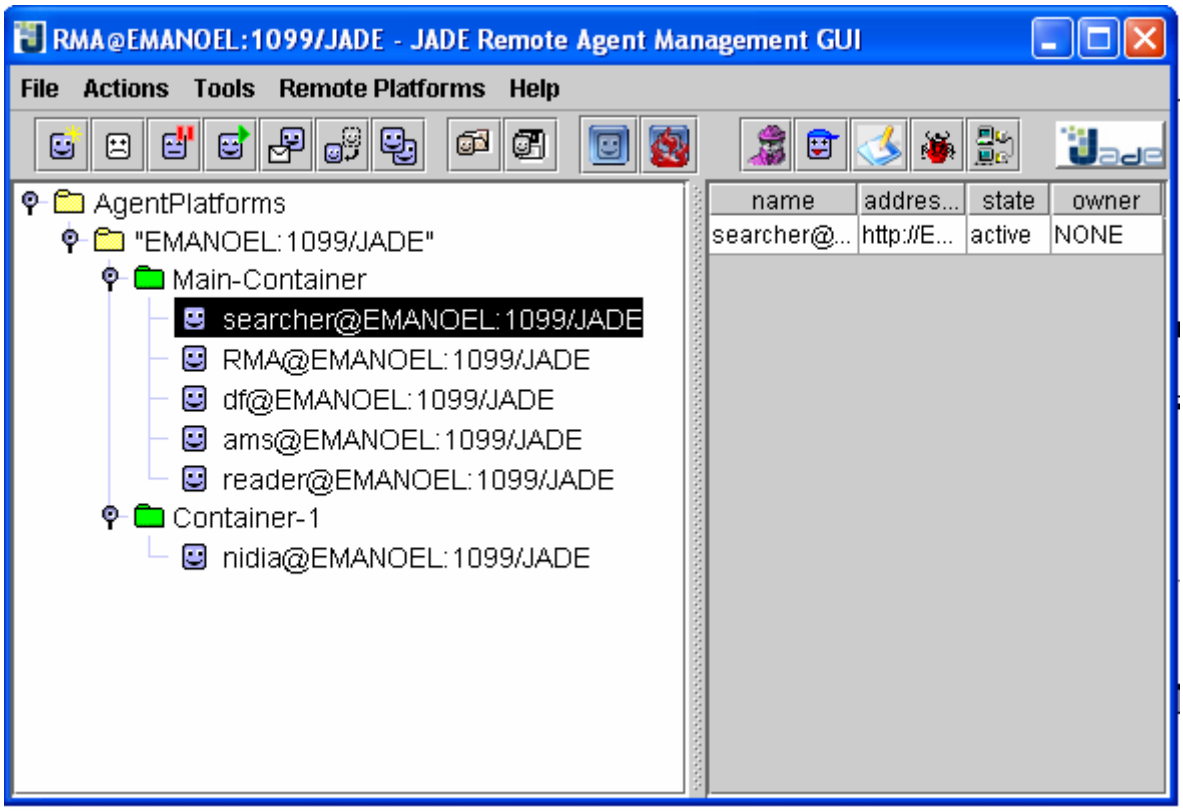

Figura 6.7. Agentes do Gerenciador de Informação, Reader e Searcher, Executando Sobre a Plataforma JADE.

A interface gráfica da plataforma JADE mostrando os agentes *SearcherAgent*, *ReaderAgent* e *NidiaAgent* em execução é ilustrada na Figura 6.7.

Durante a execução do sistema, pôde-se observar a interação do *NidiaAgent* simulando os agentes responsáveis por: detecção de ações de intrusão, Camadas de Monitoramento e Análise do NIDIA; e execução de ações de resposta, Camada de Reação. Essa interação dá-se com os agentes de busca e leitor, *SeacherAgent* e *ReaderAgent,* para atender a requisições por informações e com os agentes receptor e atualizador, *ReceiverAgent* e *UpdateAgent,* para realizar o armazenamento de informações. O agente transformador executa a conversão de dados de XML, padrão utilizado para o armazenamento, para objetos ACL e vice-versa.

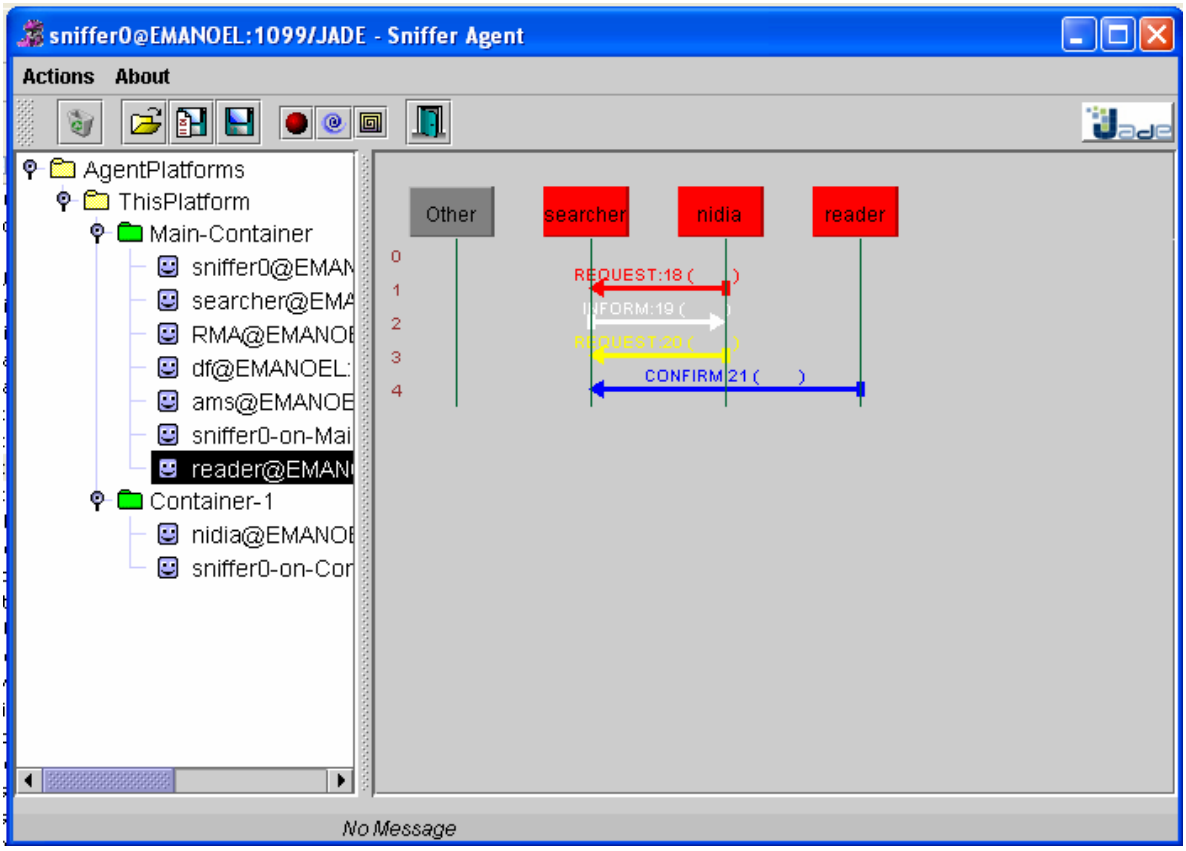

Figura 6.8. Agente Sniffer da Plataforma JADE Escutando a Troca de Mensagens entre o SeacheAgent, o ReaderAgent e o NidiaAgent.

Nas Figuras 6.8 e 6.9 mostram-se a troca de mensagens entre os agentes. Um agente utilitário da plataforma, *sniffer*, fica escutando todas as mensagens trocadas entre os agentes de um mesmo *container* ou de *container* distintos.

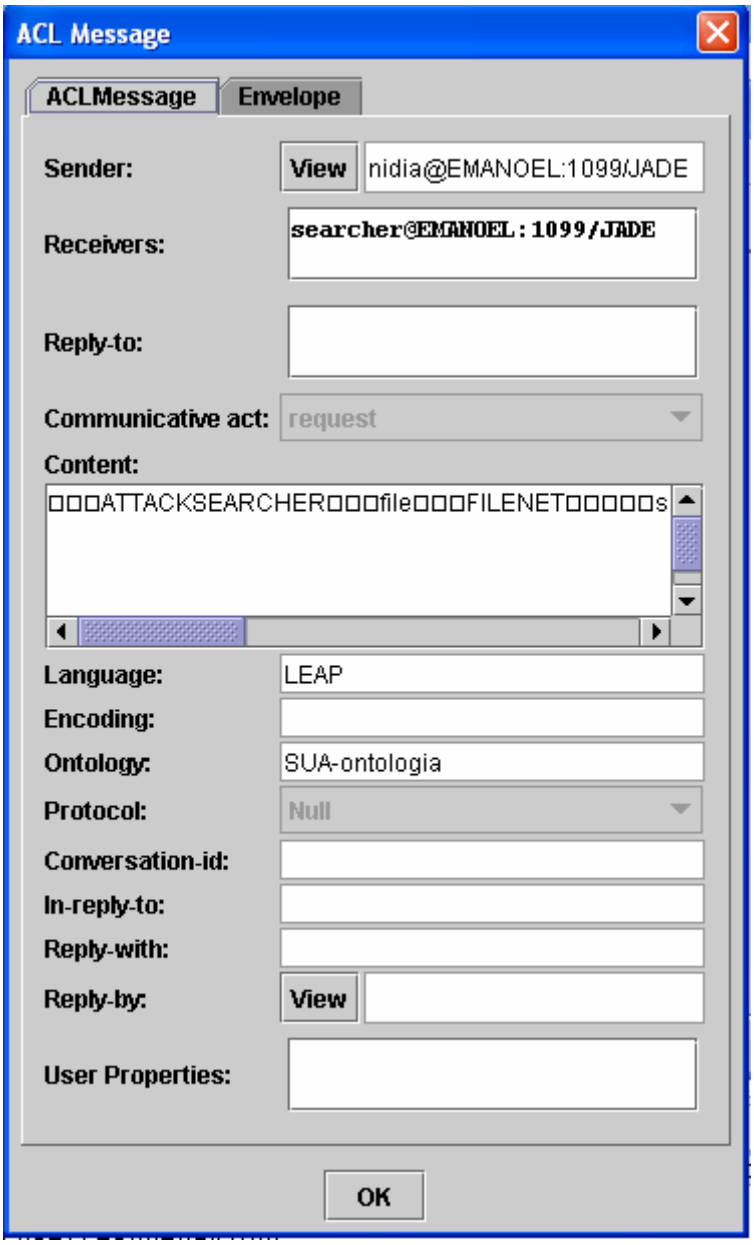

Figura 6.9. Detalhamento de Mensagem Trocada entre NidiaAgent e SeacherAgent.

O detalhamento de uma das mensagens trocadas entre o *NidiaAgent* e o *SeacherAgent* é mostrado na Figura 6.9. Na Figura 6.9, são evidenciadas informações como: Emissor, *NidiaAgent*; Receptor,

SeacherAgent; Conteúdo, ação *AttackSeacher*, com um objeto do tipo *FileNet*; Linguagem LEAP; e Ontologia, *SUA-ontologia*.

Com o intuito de melhorar o desempenho do serviço adicionou-se ao *SeacherAgent* um comportamento responsável por manter informações em memória. A idéia é manter em memória, de acesso rápido, informações que são ao mesmo tempo pequenas e muito acessadas. Para isso implementou-se no *SeacherAgent* um comportamento que estende a classe *TickerBehaviour* e executa o método *onTick()* em um dado período de tempo. Este comportamento capacita o agente de busca a interagir com o ReaderAgent e obter informações atualizadas.

As informações escolhidas para serem carregadas em memória foram às bases OKDB, uma lista de palavras-chave utilizada no método de detecção de ataques do NIDIA e a IIDB, base de dados que armazena as Assinaturas de Ataque. Sempre que possível, um *ReaderAgent* já inicializado foi utilizado para fazer as atualizações nas informações mantidas em memória pelo *SeacherAgent*, caso contrário, um *ReaderAgent* era criado para este propósito.

6.3 Conclusões

A existência de várias entidades realizando acesso aos dados de diferentes formas, além de coexistir vários modelos de armazenamento, impacta na capacidade de compartilhamento de informação e gera uma precária possibilidade de levantamento de estatísticas.

Com a utilização do Gerenciador de Informações foram introduzidas ao ambiente:

- A padronização dos dados armazenados. As informações foram todas armazenadas em XML e foram formalmente padronizadas e;
- A uniformização do acesso a dados persistentes. Assim, a coleta e compartilhamento de informações foram otimizados.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste Capítulo apresentam-se as conclusões, através dos principais pontos alcançados com esta dissertação e suas contribuições. Também, delineiam-se sugestões de futuras pesquisas.

7.1 Contribuições do Trabalho

Constitui problemática relevante na atualidade a questão da segurança digital. Inúmeras tentativas de implementação de mecanismos de proteção mais eficientes são constantemente realizadas, visto que os mecanismos utilizados não têm conseguido proporcionar o grau de segurança desejado, frustrando a expectativa de empresas e organizações que dependem e confiam nesses serviços.

Nesta perspectiva, uma ferramenta que pode ajudar a elevar o nível de segurança dos domínios sob proteção são os SDIs (Sistemas de Detecção de Intrusão), sendo, atualmente, uma das medidas de segurança mais utilizadas. No entanto, para que um SDI execute suas funções de forma eficiente faz-se prioritário que o conjunto de informações usadas para tomada de decisão esteja atualizado.

Baseado em pesquisas na área de SDI e Sistemas Multiagentes, foi proposto, nesta dissertação, como contribuição, um modelo e uma arquitetura que proporcione o compartilhamento e o gerenciamento eficiente das informações geradas por um SDI baseado na cooperação de agentes inteligentes, associadas ao uso da tecnologia de *Web Services*.

Por se tratar de um modelo baseado em agentes, este pode ser facilmente utilizado por SDIs baseados em agentes como um novo módulo dentro do sistema.

Devido à inexistência de um padrão definido para representar os dados relativos à segurança, foi proposto, também como contribuição, um modelo para a formatação e armazenamento, usando XML, das informações necessárias ao desenvolvimento das funções inerentes a um SDI. Essas informações foram agrupadas em repositórios distintos conforme suas finalidades.

Para troca de informações forenses entre SDIs, foi proposto um modelo de mensagem baseado no CAP (*Common Alerting Protocol*), utilizando *Web Services* e XML.

Como contribuição, também foram implementados os agentes responsáveis pelo armazenamento e extração de informações de forma transparente, que são: *SeacherAgent, ReaderAgent, ReceiverAgent, UpdateAgent* e *TranslatorAgent*.

7.2 Considerações Finais

Neste trabalho constatou-se a falta de padrões para o compartilhamento de informações necessárias a elevação do nível da segurança das redes de forma automática. Tanto o processo de envio e recebimento de informações dos CSIRTs (*Computer Security Incident Response Team*), quanto a interação com outros SDIs depende diretamente da intervenção do administrador. A automatização desse processo de comunicação é vital para garantir à integridade lógica das instituições, dada à velocidade com que surgem novas formas e tipos de ataques.

Porém, a simples capacidade de trocar informações não extingue o problema. A forma como as informações são tratadas internamente ao SDI

deve prover, dentre outros, requisitos como: A *Geração de Informações Uniformes*; o *Armazenamento Unificado*; e o *Acesso Transparente*.

Não foi abordada suficientemente, nesta dissertação, a questão relacionada ao desempenho do Gerenciador de Informações, pois, sabe-se que Java é uma linguagem que congrega muitas vantagens no desenvolvimento de sistemas multiagentes, entretanto apresenta problemas de desempenho. Dessa forma, este aspecto deve ser objeto de estudo detalhado em trabalhos futuros.

Quanto à situação atual do SDI NIDIA (*Network Intrusion Detection System Based on Intelligent Agent*), pesquisas recentes foram concluídas nas áreas de tolerância à falhas [55], comunicação segura com auxílio de criptografia e assinatura digital [43]. E outras estão em andamento, dentre elas, podemos citar, pesquisas que têm como objetivo propor soluções para problemas relacionados à introdução de gerenciamento remoto através de *Web Services* e MDA em SDIs [54] e segurança para componentes móveis.

7.3 Trabalhos Futuros

Como proposta para trabalhos futuros e para o aperfeiçoamento deste trabalho, apresenta-se a lista a seguir:

- Construir interface para alimentação da base de dados de estratégias;
- Implementar os agentes WSAgent e MonitorAgent responsáveis pela comunicação externa;
- Integrar o *Web Services* desenvolvido em [47] ao Módulo de Gerenciamento de Informações para que o SDI possa compartilhar suas informações internas;

- Expandir a capacidade do *TranslatorAgent*, incluindo o suporte a outros formatos de dados;
- Implementar o *ManagerAgent* e o *PropagatorAgent*, e inserir ao ambiente a capacidade de trabalhar com bases de dados replicadas;
- Apresentar uma proposta para as funções de segurança quando da utilização de *Web Services*: criptografia e assinatura digital;
- Utilizar outras plataformas de construção de agentes, e implementar o mesmo protótipo do modelo desenvolvido neste trabalho, fazendo uma comparação de desempenho entre eles;
- Realizar testes e refinar o modelo de dados para a base de estratégia, a fim de, introduzir informações referentes a dados estatísticos.

REFERÊNCIAS

[1] ANDERSON, J. P. **Computer Security Threat Monitoring and Surveillance**. Technical Report, James P. Anderson Co., Fort Washington, PA, abr. 1980.

[2] ANDERSON, D.; FRIVOLD, T. e VALDES, A. **Nextgeneration Intrusion Detection Expert System (NIDES).** Technical report, SRI-CSL-95-07, SRI International, Computer Science Lab, 1995.

[3] ARMSTRONG, Eric et al, **The J2EE 1.4 Tutorial**, Sun Microsystem, 2005.

[4] ASAKA, M.; OKAZAWA, S. e TAGUCHI, A. **A Method of Tracing Intruders by Use of Mobile Agent**. Proceedings of the 9th Annual Internetworking Conference (INET`99), San Jose, California, 1999.

[5] AUSCERT, **Australian Cert Coordination Center**. Disponível em: <http://www.auscert.org.au/>. Acesso em: 8 set. 2006.

[6] BACE, R. **Intrusion Detection**. Macmillan Tech. Pub. Indianapolis, IN, 2000.

[7] BACE, R. e MELL, P. **Intrusion Detection Systems**. National Institute of Standards and Technology Special Publication, Gaithersburg, MD, USA, 1999.

[8] BALASUBRAMANIYAN, J.; GARCIA-FERNANDEZ, J.O.; ISACOFF, D.; SPAFFORD, E.H. e ZAMBONI, D. **An architecture for intrusion detection using autonomous agents**. Technical Report COAST TR 98-05, Purdue University, Department of Computer Sciences, 1998.

[9] BAUER, D. S. e KOBLENTZ, M. E. **NIDX: An Expert System for Real-Time Network Intrusion Detection**. Proceedings of the Computer Networking Symposium, pp. 90-106, Washington, DC, abr. 1988.

[10] BISHOP, M. **Vulnerabilities Analysis: Extended Abstract**. Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection, W. Lafayette, IN, 1999

[11] CAP, Organization For The Advancement of Structured Information Standards. **Common Alerting Protocol, v. 1.1**. OASIS Standard Oct. 2005. Disponível em: <http://www.oasis-open.org/specs/index.php#capv1.1>. Acesso em: 15 set. 2006.

[12] CERT, **Coordination Center Frequently Asked Questions**. 2005. Disponível em: <http://www.cert.org/csirts/csirt_faq.html>. Acesso em 7 set. 2006.

[13] CHAVES, M. **Tutorial: Segurança na Internet**. 12° CONIP - Congresso de Informática e Inovação na Gestão Pública, junho de 2006, São Paulo – SP. Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil – Cert.br - Brasil, Junho, 2006.

[14] CIDF Working Group. **The Common Intrusion Detection Framework Architecture**. Draft CIDF, 1998.

[15] CLAUDINO, E.; ABDELOUAHAB, Z. e TEIXEIRA, M. **Management and Integration of Information in Intrusion Detection System: Data Integration System for IDS Based Multi-Agent Systems**. In: International Workshop on Interaction between Agents and Data Mining (IADM-06), Hong-Kong, China. IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-06), 2006.

[16] CLAUDINO, E.; ABDELOUAHAB, Z. e TEIXEIRA, M. **Gerenciamento das Informações de Sistemas de Detecção de Intrusão: Sistema de Integração de Dados para SDI Baseado em Sistemas Multiagentes**. In: 8o. International Symposium on Systems and Information Security (SSI 2006), São José dos Campos, SP. 2006.

[17] COSSENTINO, M. e POTTS, C. **A CASE Tool Supported Methodology for the Design of Multi-Agent Systems**. Proceedings of the International Conference on Software Engineering Research and Practice, Las Vegas, USA, 2002.

[18] CROSBIE, M. e SPAFFORD, E.H. **Defending a Computer System using Autonomous Agents**. Department of Computer Sciences, Purdue University, 1995. (Relatório Técnico CSD-TR-95-022; Coast TR 95-02). Disponível em: <http://www.cs.purdue.edu/homes/spaf/techreps/9522.os>. Acesso em 2 set. 2006.

[19] CVE. **Common Vulnerabilities and Exposures: the Standard for Information Security Vulnerability Names**. Disponivel em: <http://cve.mitre.org/>. Acesso em: 14 set. 2006.

[20] DENNING, D. E. **An Intrusion Detection Model**. IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp. 222-232, fev. 1987.

[21] 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.

[22] ECLIPSE. **Eclipse - an open development platform**. Disponível em: <http://www.eclipse.org/>. Acesso em: 22 ago. 2006.

[23] ESKIN, E. et al. **A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data**. Technical report, CUCS, 2002.

[24] ESKIN, E. et al. **Adaptive Model Generation for Intrusion Detection Systems**. Workshop on Intrusion Detection and Prevention, 7th ACM Conference on Computer Security, 2001.

[25] FEIERTAG, R.; BENZINGER, L.; RHO, S. e WU, S. **Intrusion Detection Intercomponent Adaptive Negotiation**. Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection, W. Lafayette, IN, 1999.

[26] FIPA. **The Foundation for Intelligent Physical Agents**. Disponível em: <http://www.fipa.org/>. Acesso em: 22 ago. 2006.

[27] FRINCKE, D.; TOBIN, D.; MCCONNELL, J.; MARCONI, J. e POLLA, D. **A Framework for cooperative Intrusion Detection**. Proc. 21st National Information Systems Security Conference, Arlington, VA, 1998. p. 361-373.

[28] IDEMEF, The Internet Engineering Task Force. **The Intrusion Detection Message Exchange Format**. IETF Internet-Draft, Mar. 2006.

[29] IODEF, The Internet Engineering Task Force. **The Incident Object Description Exchange Format Data Model and XML Implementation**. IETF Internet-Draft, Jun. 2006.

[30] JACKSON, K.; DUBOIS, D. e STALLINGS, C. **A Phased Approach to Network Intrusion Detection**. Proceedings of the United States Department of Energy ComputerGroup Conference, 1991.

[31] JADE. **Java Agent DEvelopment Framework**. Disponível em: <http://jade.tilab.com/>. Acesso em: 22 ago. 2006.

[32] JADE. **JADE TUTORIAL: Application-Defined Content Languages and Ontologies**. Disponível em: < http://jade.tilab.com/doc/CLOntoSupport.pdf >. Acesso em 22 ago. 2006.

[33] JAVA. **Tutorials and Online Training**. Disponivel em: <http://java.sun.com/developer/onlineTraining/>. Acesso em: 22 ago. 2006.

[34] JAVA. **The Source for Java Developers**. Disponível em: <http://java.sun.com/index.jsp>. Acesso em: 22 ago. 2006.

[35] KRÜGEL, C. e TOTH, T. **Mobile Agent Based Intrusion Detection for Dynamic Networks**, Proc. European Wireless (EW2002), 2002.

[36] KRUGEL, C. e TOTH, T. **Sparta - a security policy reinforcement tool for large networks**. In submitted to I-NetSec 01, 2001.

[37] LANE, T. e BRODLEY, C. **An Application of Machine Learning to Anomaly Detection**. Proceedings of the Twentieth National Information System SecurityConference, Baltimore, MD, 1997.

[38] LIMA, C. F. et al. **The Nidia Project Network Intrusion Detection System Based On Intelligent Agents**, Proceedings Of Tenth Latin-Ibero-American Congress On Operations Research And Systems, 2000.

[39] LIMA C.F.L. **Agentes Inteligentes para Detecção de Intrusos em Redes de Computadores**. Dissertação de Mestrado Submetida à Coordenação do Curso de Pós-Graduação em Engenharia de Eletricidade da UFMA, Maio/2001.

[40] MANDIA, K. e PROSISE, C. **Hackers Resposta e ContraAtaque**. Rio de Janeiro: Campus 2001.

[41] MUKHERJEE. B.; HEBERLEIN, T. L. e K. N. LEVITT. **Network intrusion detection**. IEEE Network. 1994.

[42] NORTHCUTT, S. **What the Hackers Know about You**. SANS Institute. SANS Institute Resources, IntrusionDetection FAQ, Hyperlink: ID FAQ, 1999.

[43] OLIVEIRA, E. J. S.; LOPES, D. e ABDELOUAHAB, Z. (2006). **Security on MASs with XML Security Specifications**. 17th International Conference on Database and Expert Systems Applications (DEXA'06), 2006. p. 5-9.

[44] OMG Green paper. **Agent Technology**. Agent Working Group OMG Document ec/2000-04-01, 2000.

[45] ONG, T.H. et al. **SNMS—Shadow Network Management System**. Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection, W. Lafayette, 1999.

[46] PASSI. **A Process for Agent Societies Specification and Implementation**. Disponível em: http://mozart.csai.unipa.it/passi/. Acesso em: 18 ago. 2006.

[47] 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.

[48] PESTANA Jr, F. A. e ABDELOUAHAB, Z. **Proposal of Model and Message Format for Sharing Information between CSIRTs and IDSs.** International Conference On Computing CIC-2005.

[49] ROESCH, M. **Snort - Lightweight Intrusion Detection for Networks**. 13th LISA Conference, 1999, Pp. 229–238 of the Proceedings. 1999.

[50] SANTOS, G. de L. F. e NASCIMENTO, E. **An Automated Response Approach for Intrusion Detection Security Enhancement**. Proceedings of VII International Conference on Software Engineering and Applications. Marina Del Rey, California, USA, 2003.

[51] SEBRING, M. M.; SHELLHOUSE, E.; HANNA, M. E. e WHITEHURST R. **A Expert systems in intrusion detection: A case study**. In Proceedings of 11th National Computer Security Conference, 1988. p. 74-81. 1988.

[52] SERVILLA, M.; HEADY, R.; LUGER, G. e MACCABE, A. **The architecture of a network level intrusion detection system**. Technical Report CS90-20, Department of Computer Science, University of New Mexico, 1990.

[53] SHERIF, J.S. e DEARMOND, T.G. **Intrusion detection systems and models.** Eleventh IEEE International Workshops on Infrastructure for Collaborative Enterprises, 2002.

[54] SILVA, M.; LOPES, D. e ABDELOUAHAB, Z. **A Remote IDS based on Multi-agent Systems, Web Services and MDA**. International Conference on Software Engineering Advances. ICSEA 2006.

[55] SIQUEIRA, L. e ABDELOUAHAB, Z. **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.

[56] SMAHA, S. E.; **Haystack: An Intrusion Detection System**. Fourth Aerospace Computer Security Applications Conference, Orlando Florida, p.37-44, dez. 1988.

[57] SPAFFORD, E. H. e ZAMBONI, D. **Intrusion Detection Using Autonomous Agents**. Computer Networks. Elsevier North-Holland, 34(4):547-570, 2000.

[58] STANIFORD-CHEN, S. **Common Intrusion Detection Framework(CIDF)**. Computer Emergency Response Team (Coordenation Center), Outubro 1998. Disponível em: <http://seclab.cs.ucdavis.edu/cidf/>. Acesso em: 8 set. 2006.

[59] STOLFO, S.J.; PRODROMIDIS, A.L.; TSELEPIS, S.; LEE, W.; FAN, D. e CHAN. P.K. **JAM: Java Agents for Meta-learning over Distributed Databases**. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA,USA, pp. 74–81. 1997.

[60] SUNDARAM, A. **An Introduction to Intrusion Detection.** Crossroads: The ACM Student Magazine, 2, 4, 1996, Hyperlink: acm.org/Crossroads, 1996.

[61] TELECOMITALIA. **Telecom Itália**. Disponível em: <http://www.telecomitalia.com/>. Acesso em: 22 ago. 2006.

[62] XINDICE. **Apache Xindece**. Disponível em: http://xml.apache.org/xindice/. Acesso em: 19 ago. 2006.

[63] WEISS, G. Multiagent Systems - **A Modern Approach to Distributed Artificial Intelligence**. The MIT Press - London, England, 1999.

[64] WYLIE, J.J.; BIGRIGG, M.W.; STRUNK, J.D.; GANGER, G.R.; KILICCOTE, H. e KHOSLA, P.K. **Survivable information storage systems**. Dept. of Electr. & Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 2000.

[65] W3C. **XML Tutorial**, 2006. Disponível em: <http://www.w3schools.com/xml/default.asp>. Acesso em: 2 out. 2006.

[66] W3C. **Extensible Markup Language (XML)**, 2006. Disponível em: <http://www.w3.org/XML>. Acesso em: 2 out. 2006.

[67] W3C. **Web Services Architecture**, 2005. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acesso em: 5 out. 2006.

[68] W3C. **Introduction to Web Services**, 2006. Disponível em: <http://www.w3schools.com/webservices/ws_intro.asp>. Acesso em: 5 out. de 2006.

[69] ZERKLE, D. e K. LEVITT. **NetKuang — A Multi-Host Configuration Vulnerability Checker.** Proceedings of the Sixth USENIX Security Symposium, San Jose, CA,1996.