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

Modelo de IDS Remoto baseado na tecnologia de Agentes, Web Services e MDA

Mauro Lopes Carvalho Silva

São Luís
2006

Mauro Lopes Carvalho Silva

Modelo de IDS Remoto baseado na tecnologia de Agentes, Web Services e MDA

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

Orientador: Zair Abdelouahab

Doutor em Ciência da Computação – UFMA

São Luís

2006

Silva, Mauro Lopes Carvalho

Modelo de IDS Remoto baseado na tecnologia de Agentes, Web Services e MDA / Mauro Lopes Carvalho Silva - São Luís, 2006.

142 f.

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

Orientador: Zair Abdelouahab.

1. Segurança. 2. Sistema de detecção de Intrusos. 3. Internet. I. Título.

CDU 004.738.5.492.3:621.3

Mauro Lopes Carvalho Silva

Modelo de IDS Remoto baseado na tecnologia de Agentes, Web Services e MDA

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

Aprovado em XX de novembro de 2006

BANCA EXAMINADORA

_____

Zair Abdelouahab
Ph.D. em Ciência da Computação - UFMA

_____

Avaliador 2
Doutor em Ciência da Computação – UFMA

_____

Avaliador 3
Doutor em Ciência da Computação - XXXX

Aos meus pais Aristéa e Emanoel Borges

Agradecimentos

A Deus pela proteção e bênçãos recebidas.

A minha mãe Aristéa, pela torcida, compreensão, carinho e preocupação.

A todos os meus irmãos, pelo apoio incondicional em toda vida.

A minha namorada Pâmela Dávalos pelo enorme carinho e compreensão em todos os momentos.

As minhas companheiras do mestrado, Lindonete e Simone, pelas incansáveis horas de estudo, inclusive aos finais de semana.

A todos os meus companheiros do Mestrado que acompanharam e contribuíram de perto ou de longe no desenvolvimento deste trabalho.

A todos os meus companheiros de trabalho pela ajuda e compreensão nos momentos difíceis de conciliação de estudo e trabalho.

Ao professor Zair Abdelouahab pela orientação.

Ao Dr. Denivaldo Lopes pelos aconselhamentos e direcionamentos fornecidos para elaboração desta dissertação.

A todos que contribuíram de alguma forma. Que Deus lhes abençoe sempre!

"Somos o que fazemos, mas somos, principalmente, o que fazemos para mudar o que somos."
Eduardo Galeano

Resumo

No atual contexto da Internet, a segurança da informação constitui-se uma preocupação permanente. Em muitos casos, a segurança da informação é vital para a manutenção e continuidade dos negócios. As organizações têm usado a Internet como um dos principais pontos para a prestação de serviços para outras organizações assim como para seus usuários finais. Podemos citar algumas organizações como Bancos, Instituições de Ensino, Administradoras de Cartões de Crédito e o Governo Federal. O uso de Políticas de Segurança associado ao uso de um conjunto de ferramentas, como Firewall, Antivírus e IDS (*Intrusion Detection System*) tem apoiado as organizações no objetivo de manter a segurança e desta forma a continuidade dos negócios. Na outra extremidade da prestação de serviços pelas organizações temos os usuários finais. A necessidade por eficácia em segurança computacional aos usuários finais tem aumentado em função do crescimento considerável na ocorrência de ataques a este tipo de usuário. Este problema cria um nicho para a pesquisa em segurança voltada ao usuário final.

Esta dissertação tem por motivação esse cenário, consistindo na proposta do modelo e a implementação de um IDS Remoto usando a tecnologia de Sistemas Multiagentes, *Web services* e MDA (*Model-Driven Architecture*). O modelo adapta e extende o NIDIA (*Network Intrusion Detection System based on Intelligent Agents*) para prover um IDS remoto na Internet. A proposta é que usuários que não têm um IDS local possam usar os serviços providos por nosso IDS Remoto. O NIDIA é um IDS cuja arquitetura consiste em um conjunto de agentes cooperativos. As funcionalidades do IDS Remoto são providas como um conjunto de serviços acessíveis na Internet através de *Web services*. O nosso modelo de IDS usa MDA para suportar o gerenciamento de metadados tais como *profiles* de configuração, *profiles* de usuários e *profiles* de serviços. A implementação do protótipo do modelo proposto e os testes realizados demonstram a viabilidade da solução. Desta forma, um exemplo ilustrativo do funcionamento do IDS Remoto é apresentado.

Palavras-chave: Segurança, Detecção de Intrusão, Usuário Final, Multiagente, *Web services*, MDA.

Abstract

In the current state of the Internet, information security presents a permanent concern. In many cases, information security is vital a maintenance and continuity of the businesses. The organizations have used the Internet as one of the main points for rendering of services for other organizations as well as for their final users. We can cite some organizations such as Banks, Institutions of Education, Administrators of Credit cards and the Federal Government. The use of Security policies associated with a set of tools such as Firewall, Antivirus and IDS (Intrusion Detection System) have helped organizations to achieve some security and thus allowing the continuity of the businesses. On the other extremity of the rendering of services for organizations we have the final users. The necessity for effectiveness in computational security to the final users has increased in function of the considerable growth on the occurrence of attacks to this type of user. This problem creates a niche for the research in security directed to the final user.

This work is motivated by the above problem. Our work consists of a proposal of a model and an implementation of a Remote IDS (Intrusion Detection System) using the technology of Multi-agent Systems, Web Services and MDA (Model-Driven Architecture). This model adapts and extends the NIDIA (Network Intrusion Detection System based on Intelligent Agents) to provide a remote IDS on the Internet. The purpose is that users that do not have a local IDS can use the services provided by a remote IDS (e.g. NIDIA). NIDIA is an IDS whose architecture consists of a set of cooperative agents. The Remote IDS functionalities are provided as a set of accessible services on the Internet through Web Services. The architecture of our IDS uses MDA to support metadata management such as profiles of configurations, profiles of users and profiles of services. The prototype of the proposed model and the tests demonstrate the viability of our solution. An illustrative example of the execution of the Remote IDS is presented.

Keywords: Security, Intrusion Detection, Final User, Multiagent, Web services, MDA.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AAFID	Autonomous Agents For Intrusion Detection
ACL	Agent Communicaton Language
BAM	Binary Association Memory
B2C	Business to Consumer
B2B	Business to Business
CIDF	Common Intrusion Detection Framework
CWM	Common Warehouse Metamodel
CGI.br	Comitê Gestor de Internet no Brasil
CERT.br	Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil
CISL	Common Intrusion Specification Language
CEO	Chief Executive Officer
COMM	Communication in the Common Intrusion Detection Framework
DARPA	Defense Advanced Research Projects Agency
DF	Directory Facilitator
DFDB	Base de Dados de Incidentes de Intrusão e Informação Forense
DoS	Denial of Service
EJB	Enterprise Java Beans
FIPA	Foundation for Intelligent Physical Agents
GIDO	Generalized Intrusion Detection Objects
HIDS	Host-Based Intrusion Detection
HTML	Hyper Text Markup Language
IIDB	Base de Dados de Padrões de Intrusos e Intrusões
IDS	Intrusion Detection System
IDA	Intrusion Detection Agent
IP	Internet Protocol
JADE	Java Agent Development Framework
JAM	Java Agents for Meta-learning
JMI	Java Metadata Interface
KQML	Knowledge Query and Manipulation Language
LSIA	Local Security Intelligent Agent
MADIDS	Mobile Agent Distributed Intrusion Detection System

MCA	Agente Controlador Principal
MDA	Model-Driven Architecture
MOF	Meta Object Facility
MDR	MetaData Repository
MLSI	Marks Left by Suspected Intruder
NIDIA	Network Intrusion Detection System based on Intelligent Agents
NIC	Núcleo de Informação e Coordenação
NIDX	Expert Network Intrusion Detection
NIDS	Network-Based Intrusion Detection
OASIS	Organization for Structured Information Standards
RADB	Base de Dados de Ações de Respostas
RMA	Remote Management Agent
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAARA	Sociedade Atualizada de Agentes de Reconhecimento de Assinaturas
SCA	System Controller Agent
SEA	System Security Evaluation Agent
SFEA	System Fault Evaluation Agent
SMA	System Monitoring Agent
SRA	System Replication Agent
SSA	System Sentinel Agent
SSL	Secure Socket Layer
STDB	Base de Dados de Estratégias
SUA	System Updating Agent
SOAP	Simple Object Access Protocol
Sparta	Security Policy Adaptation Reinforced Through Agents
TCP	Transmission Control Protocol
UFMA	Universidade Federal do Maranhão
UML	Unified Modeling Language
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
XML	eXtensible Markup Language
XMI	XML Metadata Interchange
WS-I	Web Services Interoperability Organization

W3C	World Wide Web Consortium
WSDL	Web Services Description Language

Lista de Códigos

Sumário

1 Introdução

Atualmente, as tecnologias da informação estão presentes em toda e qualquer atividade seja ela empresarial ou pessoal. Neste contexto, a comunicação, fator dos mais importantes para a realização de negócios, tomou posição de destaque. Porém, hoje em dia, não basta ter somente comunicação, é necessário que ela seja rápida, disponível e segura. Não é difícil imaginar que a não realização de uma transação, seja ela comercial ou financeira, em tempo hábil, possa comprometer a sobrevivência de uma empresa.

Desta forma, justifica-se o avanço na construção e manutenção de uma infra-estrutura de rede de computadores e telecomunicações capaz de prover a demanda de serviços disponibilizados na Internet.

Com a crescente utilização desses serviços, outros fatores foram agregados a essa infra-estrutura, como por exemplo, a segurança. Para alguns, não basta a integridade da conexão e nem sua disponibilidade, se a segurança da informação não estiver presente.

A segurança deve ser considerada como um estado obtido através da implementação e gestão de diversas práticas (mecanismos de controle de acesso, de garantia da integridade da informação, de certificação e de encriptação) que assegurem a tríade Confidencialidade[1], Disponibilidade[2] e Integridade[3] da informação.

Políticas de segurança, Planos de Continuidade de Negócios, Controle de Acessos (físicos e lógicos), a utilização de Firewalls, Antivírus, Criptografia, Sistemas de Detecção de Intrusão são alguns elementos que uma vez implantados nas organizações podem prover maior grau de segurança.

Cada um desses instrumentos traz consigo suas peculiaridades, o que deve ser levado em consideração quando da sua utilização. Por exemplo: a criptografia utilizada por uma corporação não pode ser "pesada" a ponto de comprometer a performance dos seus sistemas, inviabilizando assim sua utilização; Planos de Continuidade de Negócio devem contemplar situação de recuperação de desastres, sob pena de não cumprirem sua finalidade; Sistemas de Detecção de Intrusos devem ser configurados de forma que seus "registros" apontem para a maior quantidade de atitudes intrusivas, tendo o cuidado de evitar falsas interpretações por parte da ferramenta. Além disto, é importante fazer uso das ferramentas de segurança, mas o mais importante é saber utilizá-las conhecendo seus

[1] Confidencialidade é a propriedade de que a informação não estará disponível ou divulgada a indivíduos, entidades ou processos sem autorização.
[2] Disponibilidade é a propriedade de que a informação esteja disponível para acesso sempre que necessário.
[3] Integridade é a propriedade de que a informação não tenha sofrido alterações intencionais de fraude.

pontos positivos e negativos, procurando uma melhor forma de integrá-las para prover eficientemente a segurança de um ambiente corporativo.

No entanto, devido ao crescente uso de todo este aparato tecnológico (por exemplo, Firewalls, Antivírus, Criptografia, Sistemas de Detecção de Intrusão) e a crescente preocupação em criar diretrizes que protejam as empresas dos crescentes ataques, os invasores têm despertado para um novo filão: os usuários finais.

Na Internet, a preferência pelos usuários finais é atestada pela mudança do tipo de ataque. No período de janeiro a dezembro de 2005, o CERT.br (Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil) registrou de um total de 68.000 relatos de ataques, que somente 65 ataques foram direcionados a usuários finais. Já no período de julho a setembro de 2006, o CERT.br registrou 59.576 relatos de incidentes de segurança. Conforme a Tabela 1.1, do total de incidentes registrados, 59% representam o tipo de ataque *worms*[4] o que indica um alto uso desta prática. Já a prática de *scan*[5] vem em segundo lugar e representa 23% dos relatos. As fraudes, como falsos sites de bancos e governo eletrônico, por exemplo vêm em terceiro lugar com 18%. Estas técnicas têm sido direcionadas geralmente aos usuários finais. Já os ataques de DoS (*Denial of Service*) e os ataques a Servidores Web não chegam a representar nem 1% dos ataques registrados, sendo estas técnicas de ataques direcionadas geralmente às grandes corporações [10].

Totais Mensal e Trimestral Classificados por Tipo de Ataque.													
Mês	Total	worm (%)		DoS (%)		invasão (%)		AW (%)		scan (%)		fraude (%)	
jul	**18754**	12833	68	11	0	57	0	46	0	2823	15	2984	15
ago	**17501**	8961	51	4	0	44	0	37	0	5160	29	3295	18
set	**23321**	13499	57	4	0	26	0	38	0	5507	23	4247	18
Total	**59576**	35293	59	19	0	127	0	121	0	13490	22	10526	17

Legenda – DoS: *Denial of Service*; AW: Ataque a Servidor Web.

Tabela 1.1: Incidentes Reportados ao CERT.br - Julho a Setembro de 2006 [10]

1.1 Descrição do Problema

Os usuários finais tornaram-se o principal alvo de ataques digitais, como vírus e roubo de senhas e dados sigilosos. Ao contrário das grandes empresas, que possuem equipes especializadas em tecnologia da informação, pequenas empresas e usuários finais

[4] *Worms* é tipo de programa que se reproduz automaticamente pela internet.
[5] *Scan* são programas que vasculham a rede em busca de equipamentos conectados e que apresentem vulnerabilidades.

são mais vulneráveis a esses crimes, já que freqüentemente ignoram os riscos aos quais estão expostos. Segundo os analistas de segurança do CERT.br[6] [26], "os usuários não conhecem bem as novas tecnologias, por isso não sabem quais são seus pontos vulneráveis". Recentemente, o relatório semestral do grupo *Symantec Corporation*[7] também aponta para esta mudança. A Tabela 1.2 apresenta os principais alvos de ataques. Podemos perceber que os usuários finais se encontram em primeiro lugar nas escolhas dos atacantes [63].

Top 10 for Targeted Attacks				
Current Rank	**Previous Rank**	**Sector**	**Current Proportion of Targeted attacks**	**Previous Proportion of Targeted attacks**
1	1	Home user	86%	93%
2	2	Financial Services	14%	4%
3	6	Government	<1%	<1%
4	3	Education	<1%	2%
5	8	Information Technology	<1%	<1%
6	7	Health care	<1%	<1%
7	5	Accounting	<1%	<1%
8	10	Telecommunications	<1%	<1%
9	4	Small Business	<1%	<1%
10	14	Utilities / Energy	<1%	<1%

Tabela 1.2: Lista dos 10 principais alvos de ataque

"Os piratas virtuais consideram os computadores domésticos como a parte frágil da cadeia de segurança", comentou Artur Wong, vice-presidente sênior do *Symantec Security Response*, em resposta aos dados apresentados no relatório.

De acordo com o Relatório de Ameaça à Segurança na Internet[8], pesquisa divulgada pela *Symantec Corporation*, o Brasil é o país da América Latina com o maior número de máquinas infectadas por *bots* nocivos, ou seja, milhares de máquinas no país são controladas por invasores sem que seus donos tenham conhecimento. Na tabela de nações afetadas, o Brasil é o primeiro com 49% das máquinas infectadas.

A popular banda larga, utilizada por mais da metade dos internautas residenciais brasileiros, também pode servir para facilitar os ataques aos usuários. Como eles passam mais tempo *on-line*, quando têm acesso rápido, o tempo para ações criminosas aumenta consideravelmente.

---

[6] CERT.br, é um organismo vinculado ao CGI.br (Comitê Gestor de Internet no Brasil).
[7] Symantec Corporation é uma empresa especializada em segurança na Internet.
[8] Relatório realizado semestralmente pela *Symantec*, baseado nas ameaças identificadas por seus sistemas no mundo todo e por *honey pots* (sites-isca) usados pela companhia para estudar o comportamento dos *hackers*. O estudo foi realizado de janeiro a junho de 2006.

Os computadores domésticos são utilizados para realizar inúmeras tarefas, tais como:

- Transações financeiras, sejam estas bancárias ou mesmo compra de produtos e serviços;
- Comunicação, por exemplo, através de *e-mails* ou mensagens;
- Armazenamento de dados sejam eles pessoais ou comerciais.

Os motivos pelos quais alguém tentaria invadir um computador de um usuário final são inúmeros. Alguns destes motivos podem ser:

- Utilizar o computador do usuário final em alguma atividade ilícita, para esconder a real identidade e localização do invasor;
- Utilizar o computador do usuário final para lançar ataques contra outros computadores;
- Propagar vírus de computador;
- Furtar números de cartões de crédito e senhas bancárias;
- Furtar a senha da conta do provedor do usuário final, para acessar a Internet se fazendo passar pelo mesmo;
- Furtar dados sigilosos armazenados no computador do usuário final, como por exemplo, informações do seu Imposto de Renda;
- Furtar dados da empresa em que o usuário trabalha, em caso deste ser um CEO (*Chief Executive Officer*) de uma grande corporação e compartilhar seu computador pessoal com as atividades da empresa.

No entanto, segundo indicadores do relatório da NIC (Núcleo de Informação e Coordenação) [10], os usuários finais têm iniciado um processo de proteção de suas máquinas e informações. O número de ferramentas como antivírus, firewalls pessoais e anti-spywares tem crescido muito entre estes usuários. Quase 70% dos usuários da Internet têm antivírus, quase 19% usam firewall pessoal e 22% anti-spywares. Trata-se de percentuais bastante elevados de disseminação de medidas de segurança.

A Tabela 1.3 apresenta as estatísticas de uma pesquisa realizada pelo Instituto IPSOS sobre as Medidas de Segurança tomadas com relação ao computador por parte do usuário final brasileiro [10]. Esta pesquisa teve como espaço amostral, diversas regiões metropolitanas do Brasil, conforme pode ser percebida na legenda que acompanha a tabela.

MEDIDAS DE SEGURANÇA TOMADAS COM RELAÇÃO AO COMPUTADOR *Percentual sobre o total de usuários de internet que possuem computador*[9] Fonte: CGI.br[10]				
Percentual (%)		**Uso de antivírus**	**Uso de firewall pessoal**	**Uso de software anti-spyware**
Regiões do País	**RM SP**	70,32	30,73	25,99
	RM RJ	70,41	24,20	23,10
	RM BH	60,63	18,16	16,64
	Outras SE	73,42	21,75	20,54
	RM SAL	75,75	17,64	21,65
	RM REC	86,85	18,85	25,06
	RM FOR	77,92	23,91	37,96
	Outras NO	58,95	5,55	19,29
	RM BEL	60,76	17,09	23,42
	Outras N	62,20	5,74	20,22
	RM CUR	74,19	17,22	19,38
	RM POA	61,83	9,90	11,00
	Outras S	67,39	9,80	20,63
	DF	74,08	26,51	27,28
	Outras CO	76,46	11,79	25,90
Total		**69,76**	**19,33**	**22,09**

Legenda - *RM: Região Metropolitana; SP: São Paulo; RJ: Rio de Janeiro; BH: Belo Horizonte; SE: Sudeste; SAL: Salvador; REC: Recife;FOR: Fortaleza; NO: Nordeste; BEL: Belém; N: Norte; CUR: Curitiba; POA: Porto Alegre; S: Sul; DF: Distrito Federal; CO: Centro Oeste.*

Tabela 1.3: Medidas de Segurança tomadas com relação ao Computador [10]

No entanto, assim como em muitas empresas, o fator a ser assegurado para a segurança de usuários finais é evitar que as máquinas sejam invadidas. Porém a segurança tem se tornado uma corrida contra o relógio, onde invasores procuram brechas e as ferramentas de segurança procuram evitá-las. No entanto, quando ocorre um descompasso deste ato, é quando a invasão ocorre. Estando um invasor dentro da máquina do usuário final, qual medida deve-se tomar? É neste cenário que entra o IDS (*Intrusion Detection System*) [4]. Os IDS são sensores de detecção de intrusão que, dispostos de forma inteligente na rede, podem gerenciar os equipamentos dos clientes constantemente, monitorando violações de segurança ou utilizações indevidas originadas dentro ou fora da rede. Os IDS aumentam a proteção oferecida pelo firewall das organizações, fornecendo um sistema de aviso abrangente e em tempo real, que identifica e isola proativamente os ataques reais à segurança e ajuda a evitar gastos com interrupção das operações e possíveis perdas de receita. Na nossa visão, dispor de uma ferramenta destas para o usuário final seria de grande ajuda e justificada perante o grande número de invasões realizadas com sucesso.

Assim, um sistema de IDS se encaixaria junto às demais ferramentas de segurança para usuários finais.

[9] Base: 1.020 entrevistados que usaram internet nos últimos três meses e possuem computadores no seu domicílio. Respostas múltiplas (pesquisa realizada em agosto/setembro 2005).

[10] Pesquisa realizada pelo Instituto IPSOS.

1.2 Objetivos Gerais e Específicos da Dissertação

Esta dissertação visa propor um modelo e implementação de um IDS Remoto usando a tecnologia de Sistemas Multiagentes, Web Services e MDA (*Model-Driven Architecture*). Este modelo adapta e extende o NIDIA (*Network Intrusion Detection System based on Intelligent Agents*) [36] para fornecer um IDS Remoto que expõe seus serviços na Internet. Desta forma, usuários (usuários finais ou não) que não possuem um IDS Local podem fazer uso deste serviço de forma remota. Neste documento apresentamos a especificação dos componentes do modelo, como estes se relacionam, seu modelo de comunicação e troca de informação. A aplicabilidade do modelo é demonstrada através da implementação de seus componentes. Desta forma, pretende-se contribuir para a disseminação de pesquisas nesta área, almejando maiores avanços nas técnicas de ferramentas de uso remoto e no conjunto de ferramentas de segurança da informação. Dentro deste modelo serão enfocadas técnicas de detecção por abuso de ataques de rede, em específico, técnicas de adivinhação de senha [34]. Este tipo de ataque está sendo fortemente praticado devido à facilidade de obtenção de programas prontos para este fim e pelo crescente uso de redes de computadores nas empresas e por usuários finais. Desta forma, faremos exposição de um usuário final ao ataque, apresentaremos sua captura e contramedidas tomadas pelo sistema implementado através do modelo proposto.

Esta dissertação tem como objetivos específicos:

a. Apresentar um modelo de um IDS Remoto;
b. Apresentar os requisitos necessários para o desenvolvimento deste modelo;
c. Apresentar a especificação dos componentes que são necessários ao modelo;
d. Apresentar a implementação dos componentes do modelo, demonstrando as operações realizadas pelos mesmos;
e. Implementar um protótipo do modelo de IDS remoto extendendo o NIDIA;
f. Demonstrar o funcionamento do sistema através da simulação de um ataque ao usuário final, fazendo com que o sistema capture e tome as devidas contramedidas.

1.3 Organização da Dissertação

Esta dissertação está dividida em seis capítulos. No Primeiro Capítulo, um panorama sobre a dependência digital é apresentado. Em tal panorama, encontram-se as atividades humanas e a necessidade de se proteger os sistemas computadorizados e suas informações, seja em um ambiente empresarial ou a um usuário final.

O Capítulo 2 apresenta o referencial teórico da dissertação, expondo conceitos sobre as bases de nossa proposta. Conceitos sobre as ferramentas de detecção de intrusos, as tecnologias de Agentes, Sistemas Multiagentes e sistemas de detecção de intrusos baseados em agentes. Além destes conceitos que fazem parte de alguns IDS existentes, os conceitos sobre as tecnologias de *Web services* e MDA também são apresentados, demonstrando suas aplicabilidades aos sistemas computacionais.

O Capítulo 3 apresenta o projeto NIDIA, a sua arquitetura, sua implementação e suas diferenças perante outros sistemas de detecção de intrusos baseados em agentes.

O Capítulo 4 apresenta o Modelo de IDS Remoto que é uma extensão e adaptação do NIDIA. As características do sistema em termos de seus componentes são apresentadas. A cooperação entre os componentes é discutida, assim como a comunicação e a troca de informações. Neste capítulo, os requisitos e as soluções para que um IDS possa trabalhar de forma remota são apresentados.

O Capítulo 5 apresenta a implementação do protótipo, extendendo e adaptando o NIDIA. Sendo assim, nós mostramos: o desenvolvimento dos componentes que pertencem ao modelo; o funcionamento do sistema; utilização do sistema por um usuário final; a demonstração de um ataque ao usuário final e as contramedidas tomadas pelo sistema.

O Capítulo 6 apresenta as considerações finais da dissertação, ressaltando as contribuições da pesquisa realizada e também sugestões para trabalhos futuros.

2 Fundamentos Teóricos

Neste capítulo, os conceitos fundamentais das tecnologias que serviram de base para o desenvolvimento da nossa proposta são apresentados. Tais conceitos tiveram papel relevante no entendimento individual de cada tecnologia e na obtenção do conhecimento para melhor integrá-las. A integração destas tecnologias é o "pilar" principal do desenvolvimento da nossa proposta.

2.1 IDS (*Intrusion Detection System*)

Antes de falarmos sobre o sistema IDS é importante definir o que significa uma intrusão. Uma intrusão pode ser definida segundo SERVILLA [64], "como qualquer conjunto de ações que tentem comprometer a integridade, confidencialidade ou disponibilidade dos dados e/ou sistema". Uma forma de se defender de intrusões é fazer uso de Sistemas de Detecção de Intrusões (IDS - *Intrusion Detection System*). O primeiro conceito de IDS foi proposto por James Anderson, em 1980 [4]. Porém, somente em 1987, quando Dorothy Denning publicou um modelo de detecção de intrusão [16], foi que esta área de pesquisas em IDS começou a incrementar esforços. Em 1988, três protótipos de IDS foram criados:

- NIDX (*Expert Network Intrusion Detection*) por David Bauer e Michael Koblentz [7];
- O estudo de caso: *Expert systems in intrusion detection* por Michael Sebring [54];
- *Haystack* por Stephen Smaha [58].

Nos anos seguintes, um grande número de protótipos foi criado. Atualmente, as ferramentas de IDS são usadas em quase todas as redes de grandes corporações, governos e universidades, que se preocupam com a segurança de seu sistema. A área de detecção de intrusão é uma área comercial bem estabelecida, com grandes competidores, tais como Cisco com sua solução *Cisco Secure Intrusion Detection System*, a McAfee Inc. com sua solução McAfee *Entercept*, a VeriSign Inc. com sua solução *Managed* IDS, entre outras grandes empresas.

Muitas destas ferramentas funcionam analisando padrões do Sistema Operacional e da rede, como utilização da Unidade de Processamento Central (CPU), entrada e saída de dados, uso de memória, *logs* dos usuários, número de conexões e volume de dados trafegando em um segmento de rede. Este sistema pode formar uma base de dados sobre a

utilização do sistema, ou seja, armazenar informações durante o período de utilização. Outras já trazem base de dados com padrões previamente definidos, com possibilidade de inclusão de novos padrões.

A principal tarefa a ser desempenhada por um IDS é defender um sistema computacional pela detecção de um ataque e repelindo-o. Isto é feito através das suas três tarefas padrões:

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

A Figura 2.1 apresenta tarefas padrões realizadas por um sistema IDS [64].

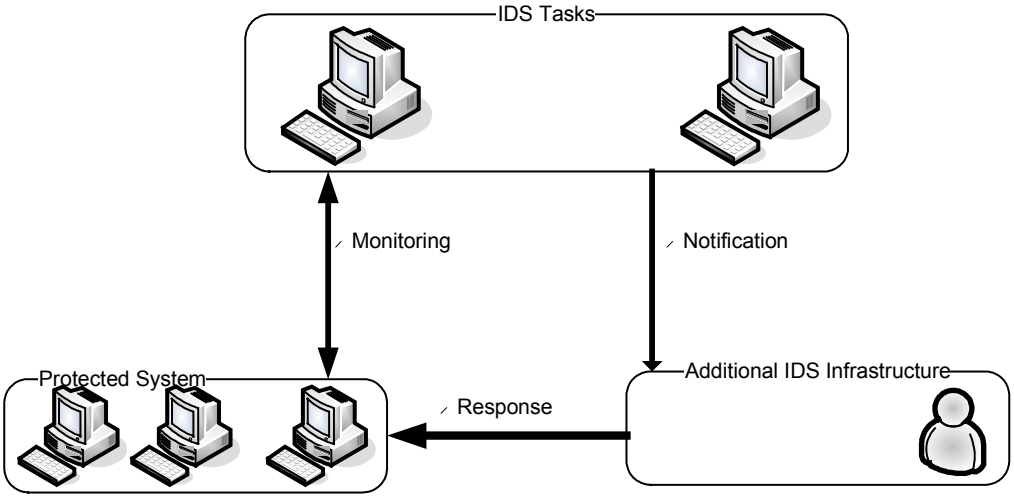

Figura 2.1: Tarefas desempenhadas pelo IDS [64]

2.1.1 Modelo Conceitual de uma Ferramenta IDS

Devido à grande variedade de sistemas de IDS, um modelo chamado CIDF (*Common Intrusion Detection Framework*) [60] foi proposto. Este modelo agrupa um conjunto de componentes que define uma ferramenta de IDS:

- Gerador de Eventos (E-boxes);
- Analisador de Eventos (A-boxes);
- Base de dados de Eventos (D-boxes);
- Unidade de Resposta (R-boxes).

A Figura 2.2 apresenta o Modelo CIDF e o relacionamento entre seus componentes.

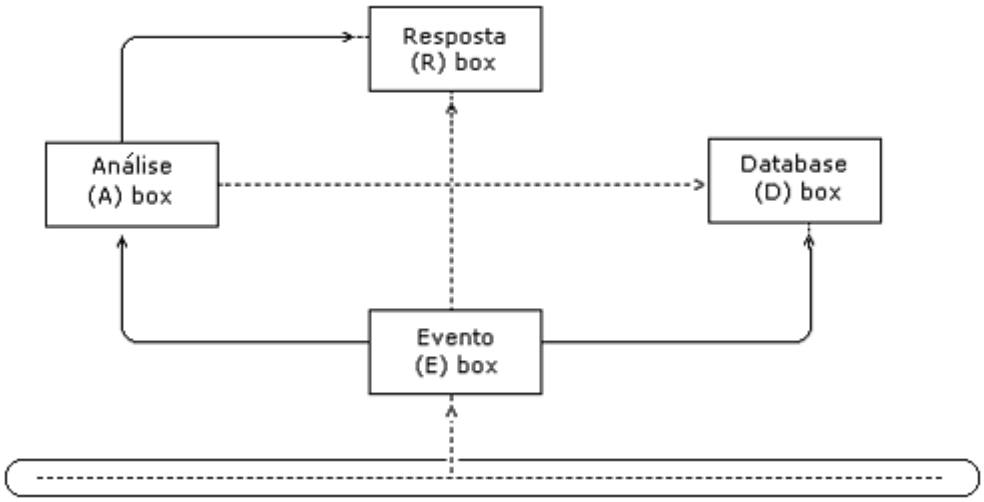

Figura 2.2: Modelo CIDF [60]

Gerador de Eventos (E-box)

A função deste componente é obter os eventos a partir do meio externo ao CIDF, ou seja, ele captura 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 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: IDS 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: IDS 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.

Segundo a padronização do CIDF, os componentes que o compõe trocam dados num formato denominado GIDO (*Generalized Intrusion Detection Objects*) [60] que podem ser representados por meio de um formato padronizado de dados conhecido como CISL (*Common Intrusion Specification Language*) [60]. Um GIDO pode ser a codificação de algum fato que ocorreu em algum momento ou a codificação de alguma conclusão obtida a partir de uma série de eventos ocorridos, ou ainda, uma instrução para disparar alguma ação.

A comunicação entre os componentes do modelo CIDF é definida por uma arquitetura de camadas:

- **Camada Gido**: Esta camada é responsável por auxiliar na interoperabilidade nos sistemas IDS. Esta camada tem sua importância principalmente devido a interoperabilidade ser um ponto significativo dentro da sua arquitetura. Os sistemas IDS devem ter um entendimento comum da semântica dos dados que eles enviam um para o outro. Assim, o formato GIDO define as semânticas para expressar eventos de interesse aos sistemas IDS;
- **Camada de Mensagem**: Esta camada garante a habilidade aos sistemas de IDS no envio de mensagens autenticadas e encriptadas de maneira confiável, através de mecanismos como firewalls e dispositivos NAT;
- **Camada de Nível de Transporte Negociado**: Devido às mensagens entre os sistemas IDS poderem ser enviadas através de vários mecanismos de transporte, o uso destes mecanismos por um sistema IDS é negociado através desta camada.

Esta negociação é feita, levando-se em consideração a necessidade de comunicação de cada componente CIDF pertencente ao sistema IDS.

Esta arquitetura em camadas garante a comunicação entre os elementos, bem como sistemas de criptografia e autenticação. Estes mecanismos estão definidos no COMM (*Communication in the Common Intrusion Detection Framework*) [60].

2.1.2 Divisão

De acordo com os esquemas de classificação de IDS [39], as técnicas utilizadas na detecção de intrusão são separadas em dois modelos diferentes: modelos de detecção de mau uso e modelos de detecção baseados em anomalia.

A detecção de mau uso, também conhecida como detecção baseada em assinaturas, consiste em identificar e utilizar padrões de ataques conhecidos para descobrir possíveis tentativas de intrusões.

A detecção baseada em anomalias classifica as atividades fora do padrão normal, identificando as que apresentam desvios de comportamento como sendo possíveis ameaças.

Os IDS podem ser classificados conforme o modelo de mau uso em:

- NIDS (*Network-Based Intrusion Detection*): Os sistemas de detecção de intrusão baseados em rede utilizam os dados coletados em segmentos de rede como dados de origem para uma auditoria;
- HIDS (*Host-Based Intrusion Detection*): Os sistemas de detecção de intrusão baseados em *host* usam os arquivos de registro de eventos do sistema como base de dados e *logs* a serem auditados.

IDS baseado em Rede

Os NIDS são considerados *sniffers* de alto nível, capturando e analisando os pacotes que passam pela rede de forma positiva, sem que os outros sistemas percebam isso. Cada pacote capturado é comparado com um conjunto de padrões de assinaturas conhecidas.

Embora os IDS baseados em rede apresentem boa eficiência em relação à detecção de ataques, eles trazem algumas questões que devem ser consideradas. Segundo NAKAMURA [39], os NIDS têm como pontos positivos:

- Um único IDS pode fornecer monitoramento para múltiplas plataformas;
- Os ataques podem ser detectados em tempo real e o administrador pode determinar rapidamente o tipo de respostas apropriado;

- Monitoram atividades suspeitas em portas conhecidas, como a porta TCP 80, que é utilizada pelo HTTP;
- Possuem capacidade de detectar não só ataques, mas também, tentativas de ataque que não tiveram sucesso.

Um dos principais problemas enfrentados pelos sistemas de detecção de intrusão em rede são as redes segmentadas. Segundo NAKAMURA [39], algumas técnicas estão sendo utilizadas para que o problema da utilização dos NIDS com *switches* seja superado.

Algumas técnicas consideradas podem ser:

- Utilização de IDS incorporado ao *switch*, porém tem limitação quanto à sua capacidade de detecção e ao seu poder de processamento;
- Utilização de porta de monitoramento (*monitor port*) dos *switches*, porém possui velocidades menores, causando a perda de pacotes monitorados;
- Utilização de sensores HIDS em conjunto com NIDS.

IDS baseado em Host

Os sistemas baseados em *host* são bastante diversificados em relação à quantidade de recursos que oferecem. Por exemplo, a maioria desses mecanismos faz monitoramento de arquivos de registro de eventos do sistema referente a atividades básicas como tentativas de *login* sem sucesso, violações de acesso e criação de novas contas de usuário. Existem também sistemas baseados em *host* que podem capturar instalações de código malicioso.

São funções de um HIDS:

- Monitorar acessos em arquivos importantes do sistema;
- Analisar modificações nos privilégios de acesso dos usuários;
- Realizar verificações da integridade dos arquivos do sistema.

Segundo ANDERSON [4], os sistemas de detecção de intrusão baseados em *host* tem pontos positivos como:

- Monitoramento detalhado das atividades específicas do sistema, como acesso a arquivos, modificação em permissões de arquivos, *logon* e *logoff* do usuário e funções dos administrados;
- Detecção de ataques que ocorrem fisicamente no servidor;
- Gera poucos falsos positivos, ou seja, os administradores recebem poucos alarmes falsos.

Os pontos negativos que devem ser considerados no HIDS são:

- Dependência do sistema operacional;
- Incapacidade de detectar ataques de rede;
- Necessidade de espaço de armazenamento adicional para os registros do sistema;
- Diminuição do desempenho no computador que está sendo monitorado.

IDS Híbrido

Além dos modelos de IDS baseados em rede e aqueles baseados em *host*, existem os sistemas de detecção de intrusão híbridos. Eles reúnem as características de NIDS e HIDS, correlacionando arquivos de registro de eventos e informações de sistema de tráfego de rede. O IDS Híbrido consiste em combinar os aspectos positivos do HIDS e do NIDS, para que a detecção de intrusão se torne eficaz.

Esses sistemas operam como se fossem NIDS, capturando e processando pacotes do tráfego da rede e detectando e reagindo a ataques. Porém, efetuam esse processo como HIDS, ou seja, processando os pacotes endereçados ao próprio sistema. Dessa maneira, é possível resolver o problema de desempenho dos IDS baseados em rede. Entretanto, o problema da escalabilidade em sistemas baseados em *host* continua, sendo que um IDS híbrido deve ser instalado em cada equipamento monitorado.

Para a proteção dos recursos de uma organização, acredita-se que a melhor estratégia ao se tratar de sistemas de detecção de intrusão, é usar os dois tipos de modelos de IDS. No cenário onde uma organização possui servidores de Internet importantes, o NIDS poderá detectar alguns ataques e o HIDS outros, garantindo uma melhor prevenção contra ataques.

2.2 Agentes e Sistemas Multiagentes

Não há um conceito unânime e uniforme para o termo “agente”. O que existe na realidade é uma convergência para características fundamentais que uma “peça de software” deve possuir para ser considerada um agente.

Uma definição bem completa é dada por [8], que descreve um agente como sendo uma entidade de software que funciona de forma contínua e autônoma em um ambiente. Segundo [8], um agente também deve ser capaz de perceber e atuar no seu ambiente, de forma flexível e inteligente, sem requerer intervenção ou orientação humana constantes. Idealmente, um agente deve aprender através da experiência, comunicar-se e cooperar com

outros agentes que porventura co-existam no mesmo ambiente. Ainda, um agente pode mover-se de um local para outro a fim de satisfazer os seus objetivos.

2.2.1 Categoria de Agentes

Para uma melhor compreensão da aplicabilidade da tecnologia de agentes, é interessante falar sobre os diversos tipos de agentes e suas mais variadas diferenças para que tenhamos uma melhor noção de utilidade no emprego de agentes. É possível fazer uma classificação de agentes de acordo com vários aspectos como quanto à mobilidade, quanto ao relacionamento inter-agentes e quanto à capacidade de raciocínio.

Assim, os agentes podem ser classificados em [41]:

- **Agentes Móveis:** são agentes que tem a mobilidade como característica principal, isto é, uma capacidade de mover-se seja por uma rede interna local (*intranet*) ou até mesmo, pela Internet, transportando-se pelas plataformas levando dados e códigos. Seu uso tem crescido devido alguns fatores como a heterogeneidade cada vez maior das redes e o grande auxílio que agentes podem fornecer em tomadas de decisões baseadas em grandes quantidades de informação;
- **Agentes situados ou estacionários:** são aqueles opostos aos móveis, isto é, são fixos em um mesmo ambiente e/ou plataforma. Não se movimentam em uma rede e muito menos na Internet;
- **Agentes Competitivos:** são agentes que "competem" entre si para a realização de seus objetivos ou tarefas, ou seja, não há colaboração entre os agentes;
- **Agentes Coordenados ou Colaborativos:** agentes com a finalidade de alcançar um objetivo maior realizam tarefas específicas, porém coordenando-as entre si de forma que suas atividades se completem;
- **Agentes Reativos:** é um agente que reage a estímulos sem ter memória do que já foi realizado no passado e nem previsão da ação a ser tomada no futuro. Não tem representação do seu ambiente ou de outros agentes e são incapazes de prever e antecipar ações;
- **Agentes Cognitivos:** esses agentes, ao contrário dos agentes reativos, podem raciocinar sobre as ações tomadas no passado e planejar ações a serem tomadas no futuro. Em outras palavras, um agente cognitivo é capaz de "resolver" problemas por ele mesmo. Ele tem, objetivos e planos explícitos os quais permitem atingir seu objetivo final. Sua representação interna e seus mecanismos de inferência lhe

permitem atuar independentemente dos outros agentes e lhe dão uma grande flexibilidade na forma de expressão de seu comportamento.

2.2.2 Sistemas Multiagentes

A abordagem multiagentes tem sido apontada por muitos pesquisadores como adequada ao desenvolvimento de sistemas de software complexos devido ao seu alto nível de abstração, o que torna mais conveniente a análise e o projeto de tais sistemas [76] [1].

Um único agente pode requerer muito conhecimento para resolver problemas complexos. Em alguns casos, o problema é tão complexo que um agente não pode, por ele mesmo, resolvê-lo [9]. Além disso, muitos problemas têm características distribuídas e, portanto, necessitam de unidades distribuídas de conhecimento para a sua resolução. Portanto, seria interessante a existência de unidades independentes que resolvesse cada questão individualmente e que juntos resolvessem o problema todo, ou seja, um sistema composto por mais de um agente que trabalham juntos para alcançarem um objetivo comum. Esses sistemas são denominados Sistemas Multiagentes (SMA).

Uma vantagem dos sistemas multiagentes, segundo [9], é o fato de eles permitirem a integração de agentes já existentes. Isso faz com que a solução do problema não demande o desenvolvimento de novos e especializados agentes, ou seja, o conhecimento dos agentes existentes pode ser utilizado para resolver determinado problema.

O conhecimento especializado de cada agente e seu uso no desenvolvimento de uma estratégia comum de solução só podem ser utilizados por outro agente através de mecanismos de comunicação e cooperação [9].

Deste modo, a resolução de problemas através de SMA só é possível quando os agentes têm capacidades de se comunicarem e de cooperarem uns com os outros. Além disso, é necessária alguma forma de coordenação do comportamento dos agentes.

2.2.3 Comunicação entre Agentes

A comunicação é o processo pelo qual são trocadas informações entre os agentes. Segundo [9], os métodos de comunicação podem ser divididos em dois grupos: quadro de avisos e trocas de mensagens.

Quadro de Avisos

Os sistemas baseados em quadro de avisos, também conhecidos como sistemas de quadro negro, mantêm uma área de trabalho comum nas quais os agentes podem trocar informações, dados e conhecimentos.

Uma ação de comunicação é iniciada quando um agente escreve alguma coisa no quadro negro. Os agentes podem acessar o quadro negro a qualquer momento e verificar se alguma informação foi modificada ou adicionada.

Troca de Mensagens

A comunicação usando mensagens forma uma base flexível para a implementação de estratégias de coordenação complexas.

Um agente, denominado emissor, envia uma mensagem a um outro agente, o receptor. Em contraste com os sistemas de quadro negro, as informações são trocadas diretamente entre os agentes.

Em [19], a troca de mensagens é classificada em:

- **Ponto-a-ponto:** As mensagens são enviadas para um agente específico que deve ser conhecido pelo emissor. Uma grande vantagem da abordagem ponto-a-ponto é que o agente emissor sempre sabe para onde uma mensagem está sendo enviada, o que permite a fácil introdução de controles de segurança;
- ***Broadcast:*** A mensagem é enviada para todos os agentes da sociedade. Na abordagem *broadcast*, um agente pode ser substituído por outro equivalente e o processamento do sistema será inalterado. A passagem de mensagem por *broadcast* não é segura, já que qualquer agente pode examinar o conteúdo de qualquer mensagem;
- ***Multicast:*** A sociedade de agentes é dividida em grupos. Desta forma, as mensagens podem ser enviadas somente aos agentes de determinados grupos.

Para que o mecanismo de troca de mensagens possa ser utilizado na implementação de estratégias de cooperação é preciso que seja definido um protocolo de comunicação que especifique o exato processo de comunicação, um formato para as mensagens e a linguagem de comunicação escolhida. Nesse aspecto, as linguagens de comunicação de agentes são um recurso poderoso e importante para o processo de comunicação entre agentes.

2.2.4 Linguagens de Comunicação entre Agentes

O mecanismo de troca de mensagens necessita da definição de um protocolo de comunicação que especifique o exato processo de comunicação, um formato para as mensagens e uma linguagem de comunicação. Esses requisitos nos levam às Linguagens de Comunicação de Agentes (ACL) e aos serviços de transportes de mensagens.

Uma ACL determina um meio através do qual uma atitude (por exemplo, afirmação, requisição e consulta) a respeito do conteúdo da troca de conhecimentos é comunicada [19].

KQML

A KQML (*Knowledge Query and Manipulation Language*) é uma linguagem e um protocolo para troca de informações e conhecimento entre agentes [15]. Na prática, até o ano de 2000, quando foi lançada a primeira implementação da FIPA-ACL pela FIPA (*Foundation for Intelligent Physical Agents*), ela era a única linguagem padronizada e com implementação disponível [23] e, portanto, dominava a área de comunicação entre agentes.

A KQML foi muito criticada na década de 1990. Segundo [35], a maior parte das críticas era dirigida às imprecisões na semântica de sua gramática, tal como definida pela DARPA(*Defense Advanced Research Projects Agency*). Essas imprecisões, atribuídas também ao fato dela ser informal, mas principalmente ao fato da própria KQML ter sido desenvolvida sem as idéias de sistemas multiagentes, teriam dificultado a construção deste tipo de sistemas de forma a serem compatíveis e interoperáveis [35]. A Figura 2.3 apresenta um trecho de uma mensagem KQML. Nesta Figura, temos um agente comprador (:*sender*) perguntando (:*content*) a um agente vendedor (:*receiver*) o preço de uma mercadoria. O agente vendedor dá como resposta (:*reply-with*) o preço da mercadoria.

```
(ask-one
      :sender agente-comprador
      :receiver agente-vendedor
      :content (PRECO MERCADORIA ?preco)
      :reply-with preco-mercadoria
      :language LPROLOG
      :ontology compra-e-venda)
```

Figura 2.3: Mensagem KQML (ask-one)

FIPA-ACL

A FIPA-ACL [21] é uma linguagem de comunicação de agentes que vem ganhando espaço. Ela foi apresentada inicialmente como uma alternativa bem fundamentada para a KQML. Porém, apesar de ter sido proposta em 1997, foi somente a partir do ano 2000, com os lançamentos do padrão FIPA 2000 e da plataforma FIPA-OS, que ela passou a ter uma implementação disponível e a competir com a KQML.

A estrutura das mensagens FIPA-ACL é muito parecida com a estrutura das mensagens KQML. Cada mensagem é composta por um ato comunicativo (*performativ*e) e um conjunto de parâmetros. O campo que representa o ato comunicativo é o único que é obrigatório pois este campo indica qual ato de comunicação está sendo transportado pela mensagem. Porém, a grande maioria das mensagens contém um emissor, um receptor e um campo de conteúdo. Os parâmetros podem ser alocados em qualquer posição dentro da mensagem. Além disso, as implementações são livres para definirem novos parâmetros desde que seja utilizado o prefixo "x-". As Figuras 2.4 e 2.5 apresentam exemplos de mensagens FIPA-ACL.

Na Figura 2.4 , o agente *i* pergunta se o agente *j* está registrado no domínio *d1*:

```
(query-if
      :sender (agent-identifier :name i)
      :receiver (set (agent-identitfier :name j))
      :content "((registered (server d1) (agent j)))"
      :reply-with r09
      )
```

Figura 2.4: Mensagem FIPA-ACL (query-if)

Na Figura 2.5 o agente *j* responde que não:

```
(inform
      :sender (agent-identifier :name j)
      :receiver (set (agent-identifier :name i))
      :content "((not (registered (server d1) (agent j))))"
      :in-reply-to r09
      )
```

Figura 2.5: Mensagem FIPA-ACL (inform)

2.2.5 Plataformas de Sistemas Multiagentes

A plataforma de Agentes é uma arquitetura tecnológica que provê um ambiente no qual agentes podem existir ativamente e cooperar para alcançar seus objetivos [67]. Uma plataforma de agentes pode adicionalmente suportar o desenvolvimento de agentes e aplicações baseadas em agentes.

Desta forma, uma plataforma de Sistemas Multiagentes deve [67]:

- Prover uma infra-estrutura especificando protocolos de comunicação e interação;
- Ser geralmente aberta e não possuir apenas um projetista – não é centralizado;
- Conter agentes que são autônomos e distribuídos, e competitivos ou cooperativos.

Existem atualmente várias plataformas para auxiliar o desenvolvimento de Sistemas Multiagentes. Essas plataformas disponibilizam APIs (*Application Program Interface*) para a criação dos agentes e ambientes para execução do sistema construído.

Um importante ponto na escolha de uma plataforma de Sistemas Multiagentes, é que esta esteja de acordo com os padrões definidos pela FIPA.

No site da FIPA [24], uma lista de plataformas que estão de acordo com a sua especificação é disponibilizada. Nesta lista, encontramos as seguintes plataformas: ZEUS [25], FIPA-OS [2], JADE [65], Agent Development Kit [75], JACK Intelligent Agents [3], Grasshopper [29] e April Agent Platform [40].

Segue abaixo uma breve descrição de algumas plataformas existentes e que pertencem à lista de plataformas padrões FIPA:

ZEUS

ZEUS é um *framework open source* para o desenvolvimento de agentes colaborativos ou cooperativos [25]. O ZEUS foi criado como parte do projeto de pesquisa Midas e Agentcities desenvolvidos pela British Telcom Intelligent Agent Research. A última versão do ZEUS é datada de 23 de maio de 2001 e a *homepage* do projeto recebeu sua última atualização em janeiro de 2002. Os agentes desenvolvidos pela ferramenta são baseados em JAVA. Seu ambiente está dividido em três componentes:

- *The Agent Component Library*: é uma coleção de classes que formam o bloco de construção de um agente Zeus. O conteúdo desta biblioteca é direcionado para questões de comunicação, ontologia e interação social. Ela

é composta pelos elementos *Communication*, *Social Interaction*, *Data Structures*, *Planning and Scheduling* e *User Interface*;

- *The Agent Building Tools*: é um conjunto de ferramentas que dão suporte no desenvolvimento de um agente Zeus. Este conjunto de ferramentas é direcionado à "programação" do agente Zeus. O *Visual Editors*, *Code Generator* e *Legacy System* API fazem parte deste conjunto de ferramentas;
- *The Visualisation Tools*: é um conjunto de ferramentas utilitárias que dão suporte na criação e acompanhamento do ciclo de vida de um agente Zeus. O *Name Server*, *Facilitator*, *Society Viwer*, *Agent Viwer*, *Control Tool*, *Reports Tool* e *Statistics Tool* fazem parte deste conjunto de ferramentas.

FIPA – OS

A FIPA-OS é uma plataforma para agentes originada de pesquisas da Nortel Networks Harlow Laboratories do Reino Unido [26]. O FIPA-OS é uma implementação de código aberto, feita em Java, que possui elementos obrigatórios pertencentes ao padrão FIPA para a interoperabilidade de agentes. O principal objetivo do FIPA-OS é reduzir as atuais barreiras na adoção da tecnologia FIPA através do suplemento de documentos e especificações técnicas com código aberto gerenciável.

A FIPA-OS é distribuída em duas versões:

- *Standard* FIPA-OS;
- MicroFIPA-OS (específico para dispositivos portáteis).

JADE

O JADE [65] é um *framework* de software para o desenvolvimento de sistemas multiagentes, que segue as padronizações da FIPA. O objetivo desse *framework* é simplificar o desenvolvimento de SMAs (Sistemas MultiAgentes) ao mesmo tempo em que assegura a utilização do padrão da FIPA. O JADE é, na verdade, um *middleware* que implementa uma plataforma de agentes e um software de desenvolvimento, facilitando o desenvolvimento e o gerenciamento de agentes. A plataforma pode ser distribuída por várias máquinas (que não precisam compartilhar o mesmo sistema operacional) e sua configuração pode ser controlada por uma interface gráfica (GUI) remota.

JADE é completamente implementado na linguagem Java, é *open-source* e possui as seguintes características:

- É escalável, podendo ser expandido com o adicionamento de novos agentes ao sistema;
- Suporta mobilidade de código e estado dos agentes, oferecendo uma interface gráfica de usuário (GUI), para permitir o controle de vários agentes e plataformas ao mesmo tempo;
- Permite implementação de aplicações multi-domínio, ou seja, a interface simplifica o registro de agentes em um ou mais domínios;
- Interoperabilidade, já que obedece às especificações da FIPA;
- Uniformidade e portabilidade, fornecendo um conjunto homogêneo de APIs que são independentes das características de rede e da versão de Java;
- É fácil de usar, já que auxilia o usuário, de forma transparente, em todos os aspectos da comunicação entre agentes. Dessa forma, programadores não necessitam conhecer todas as facilidades fornecidas pelo *midleware*;
- A segurança é preservada já que existem mecanismos para autenticar e verificar os "direitos" dos agentes;
- Fornece gerência do ciclo de vida do agente.

O JADE dispõe, atualmente, das seguintes ferramentas:

- **Agente de Monitoramento Remoto (RMA - Remote Monitoring Agent)**

Responsável pela administração e controle da plataforma, sendo que mais de um GUI pode ser ativado (o JADE mantém coerência entre os RMAs através de envio de *multicasting* entre eles);

- **Agente Dummy**

Agente de monitoramento e Debug para enviar, receber e armazenar mensagens ACL. Útil para testar conversações;

- **Agente Sniffer**

Agente para debug, permitindo escutar e salvar em arquivo a comunicação entre agentes. O agente *Sniffer* trabalha interceptando as mensagens ACL em trânsito e exibindo uma notação gráfica muito semelhante ao diagrama de seqüência da UML;

- **Agente Introspector**

Agente utilizado para debug, que permite monitorar o ciclo de vida dos agentes, suas mensagens ACL trocadas e os comportamentos em execução;

- **Agente SocketProxyAgent**

Agente que age como uma porta bidirecional entre a plataforma e uma conexão TCP/IP;

- **DF GUI**

Interface de usuário que permite visualizar agentes registrados e registrar novos agentes e serviços.

2.3 IDS baseado em Agentes

O uso dos Agentes de Software no desenvolvimento de sistemas IDS, especialmente, quando os agentes são equipados com mobilidade, proporciona a introdução das características da tecnologia de agentes no projeto de IDS. Assim, a utilização de agentes móveis é de grande auxílio na realização da detecção de intrusão distribuída.

Um IDS quando baseado na tecnologia de agentes consiste de vários agentes trabalhando juntos. Visto que os ataques mudam dinamicamente, as assinaturas também mudam. Desta forma, os agentes devem estar habilitados a aprender novas assinaturas ou detectar tráfego anormal resultando de novos ataques [25].

Este novo enfoque tem como objeto o desenvolvimento de arquiteturas distribuídas, onde sensores (baseados em *host* ou rede) coletam dados, pré-processam estes e enviam para uma estação centralizadora a qual está habilitada a processar estes dados. Tais como as arquiteturas Cliente-Servidor estes também sofrem das seguintes deficiências [32]:

- Um analisador central é um ponto crítico de falha. Quando um intruso manuseia a ação de entrada e saída dos dados (como um ataque de Negação de Serviço), toda a rede perde esta proteção;
- Quando toda a informação é processada em um único ponto de localização, o sistema não é escalável. A limitação da capacidade de análise das unidades de processamento em uma grande rede pode levar a um excessivo tráfego sobre a rede;
- Dificuldade em aplicar reconfigurações às estações com sensores. Usualmente, todo o sistema tem de ser reiniciado após uma modificação.

Desta forma alguns projetos foram desenvolvidos com o objetivo de demonstrar o uso e a capacidade da associação de agentes e IDS, tendo como preocupação principal sanar as dificuldades apresentadas. Os projetos visam usar as vantagens da abordagem de agentes, tais como: reconfiguração e atualização dos agentes sem causar problemas ao

restante do sistema, a capacidade de usar um agente ou um grupo de agentes pode realizar diferentes funções simples ou criar uma sociedade de agentes para desempenhar atividades complexas através da troca de informações entre os agentes da sociedade.

2.3.1 Projetos existentes

A seguir é feita uma análise de alguns Sistemas de Detecção de Intrusão baseados em agentes. Suas principais características, seus métodos de análise e de detecção serão destacadas. Os projetos apresentados são: Sparta (*Security Policy Adaptation Reinforced Through Agents*) [33], AAFID (*Autonomous Agents for Intrusion Detection*) [6], JAM (*Java Agents for Meta-learning*) [59], Hummingbird [22], IDA (*Intrusion Detection Agent System*) [5].

Sparta

Sparta [33] é o nome de um projeto patrocinado pela União Européia. É um sistema cujo objetivo principal é detectar violações de segurança em ambientes de rede heterogêneos. A arquitetura de Sparta foi projetada para alcançar um grande número de aplicações, indo do gerenciamento da rede até a detecção de intrusão.

O Sparta é um *framework* arquitetural o qual ajuda a identificar e relatar eventos interessantes que podem ocorrer em diferentes *hosts* de uma rede. Além da detecção de padrões de interesse, o Sparta pode também ser utilizado para coletar dados estatísticos de certos eventos, tais como valores extremos de um evento ou a soma destes.

O Sistema monitora eventos locais em um número de *hosts* conectados a rede.

Cada *host* tem pelo menos um gerador de eventos local, um componente de armazenamento e uma plataforma de agentes móveis instalados. A geração de eventos locais é feita por sensores que monitoram a ocorrência de eventos na rede ou no próprio *host*.

O exato tipo de evento e seus atributos são determinados pela necessidade da aplicação. Na configuração atual, o SNORT [53] está sendo usado para extrair eventos interessantes do tráfego da rede. Os eventos são armazenados em um banco de dados local para após serem analisados pelos agentes. Um subsistema de agentes móveis é responsável por fornecer um sistema de comunicação[11] e por prover um ambiente de execução para os

[11] O sistema de comunicação permite mover o estado e o código dos agentes entre diferentes *hosts*.

agentes. Adicionalmente, o sistema tem de fornecer proteção contra os riscos de segurança envolvidos quando da utilização de códigos móveis.

AAFID

O projeto AAFID [6] é um projeto flexível, cuja infra-estrutura do IDS distribuído tem algumas similaridades ao design do MAIDS [57] (*Mobile Agent Intrusion Detection System*). O AAFID é um IDS desenvolvido usando agentes Perl para uma mais rápida prototipação e compatibilidade através das plataformas. A comunicação entre agentes não segue um padrão definido, ou seja, não há um trâmite de informações a ser seguido. Este projeto também tem sua análise de detecção baseada em agentes. O diferencial do projeto está no uso de algoritmos de aprendizagem, *data warehouse* e agentes móveis. Este sistema é implementado em Java.

JAM

O projeto JAM [59] da Columbia University usa de agentes inteligentes e distribuídos escritos em Java e *data mining* para aprender modelos de fraude e comportamentos intrusivos que podem ser compartilhados entre organizações. Este projeto difere na concentração da detecção de intrusão aplicada a uma organização, usando de *data warehouse* para agregar dados dos seus departamentos e prover uma visão do lado organizacional das informações de segurança.

Hummingbird

A Universidade de Idaho desenvolveu o projeto Hummingbird [22]. O sistema Hummingbird é um sistema distribuído para gerenciar o uso impróprio dos dados. O sistema Hummingbird emprega um conjunto de agentes Hummer, cada agente Hummer é disposto em um único *host* ou em um conjunto de *hosts*.

Os agentes Hummer interagem entre si através de agentes específicos. São eles:

- Agente *Manager*: estes agentes podem transmitir comandos para os agentes *Subordinate*; comandos podem ser solicitações para iniciar/parar a captura de dados ou enviar/receber dados para outros agentes;
- Agente *Subordinate*: estes agentes recebem os comandos e solicitam auxílio dos agentes *Peer* para executar os comandos recebidos;
- Agente *Peer*: estes agentes decidem como honrar as solicitações feitas pelos agentes *Manager*.

O sistema Hummingbird usa a tecnologia de agentes de forma limitada, pois os agentes não são autônomos e nem são móveis. Apenas a coleta dos dados é distribuída, o controle permanece centralizado. A ênfase é deslocada para segurança no compartilhamento dos dados entre sites tendo diferentes domínios de segurança. O projeto Hummingbird não implementa novas características de segurança pra proteger a rede e a si mesmo, para tal finalidade este utiliza a solução do sistema Kerberos [38].

IDA

A *Information-technology Promotion Agency* (IPA), no Japão, está desenvolvendo um IDS chamado *Intrusion Detection Agent* (IDA) [5]. O IDA é um IDS baseado em multi-hosts. Realizando a análise de todas as atividades de seus usuários, IDA trabalha observando eventos específicos que podem identificar as intrusões, referenciando-se a estes eventos como MLSI[12] (*Marks Left by Suspected Intruder*). Caso um MLSI seja encontrado, IDA captura informações relacionadas ao MLSI, analisa estas informações e decide se o que aconteceu é ou não uma intrusão.

O IDA conta com agentes móveis para detectar intrusos através de vários *hosts* envolvidos em uma intrusão. A sua arquitetura é hierárquica, com um gerenciador central na raiz e uma variedade de agentes nos demais níveis.

O sistema consiste em:

- **Um gerente centralizado**: o gerente analisa as informações coletadas e depositadas no *bulletin board* pelos *information-gathering agent*;
- **Sensores estáticos**: estes agentes residem em um nó que procura por MLSIs. Se um sensor detecta um evento suspeito, reporta a ocorrência do mesmo ao gerente que envia um *tracing agent* ao *host*;
- ***Bulletin boards***: são depósitos de informações usadas pelo gerente centralizado na tomada de decisões;
- ***Message boards***: este componente é usado em um *host* por agentes para troca de informações localmente;
- ***Tracing agents*** : este agente ativa um *information-gathering agent* para coletar informações sobre o evento suspeito. O *tracing agent* investiga o ponto de origem do evento suspeito, movendo-se ao longo do rastro e

[12] O IDA define MLSI como sendo os dados capturados durante a sua fase de observação de eventos.

ativando novos *information-gathering agents* em cada *host* que chega ao longo da rota

- ***Information-gathering agents***: realiza a coleta das informações do sistema e registra as informações colhidas no *bulletin board*.

2.4 Web Services

Atualmente, a Internet é o principal ambiente distribuído para a integração de sistemas e aplicações, visto que a Internet é estruturada sobre um conjunto de padrões aceitos universalmente. No entanto, a Internet por si só, não oferece todos os mecanismos necessários para que a integração de sistemas e aplicações seja possível.

Neste contexto, os *Web Services* surgem, como a tecnologia que permite a interoperabilidade universal, utilizando os padrões de Internet e, conseqüentemente, favorecendo a integração de sistemas e aplicações, especialmente nos domínios B2C (*Business to Consumer*) e B2B (*Business to Business*).

Os *Web services* são serviços oferecidos por uma aplicação para outras aplicações via Web [68]. Os clientes desses serviços podem agregá-los para formar um software final, que possibilita transações comerciais ou criar um novo *Web Service* [61]. Uma empresa pode ser fornecedora de *Web Services* e também consumidora de outros serviços.

A tecnologia dos *Web Services* não é baseada em nenhuma linguagem ou tecnologia de programação. É possível implementar um *Web Service* em qualquer linguagem que forneça mecanismos para comunicação com os requisitos e parâmetros definidos na sua arquitetura.

A Arquitetura de *Web Services* começou a ser regulamentada pela W3C *(World Wide Web Consortium)* [68] em 2002 e segue os seguintes parâmetros:

- *Web Services Architecture*: referência de procedimento para o modelo *Web Services* [69];
- Tecnologias de apoio aos *Web Services*: XML (*eXtensible Markup Language*) [74] e protocolo SOAP (*Simple Object Access Protocol*) [24];
- *Web Services Description Language* (WSDL): descrição para interface do *Web Services* [11];
- UDDI (*Universal Description*, *Discovery and Integration*): registro para publicar e descobrir provedores, serviços Web disponibilizados e interfaces técnicas que consumidores utilizam para interagir com os serviços [42].

2.4.1 *Web Services Architecture*

O objetivo do *Web Services Architecture* [69] da *W3C*, chamado também de *WSA*, é fornecer meios padronizados de interação entre softwares de diferentes aplicações, independente das várias plataformas e/ou estruturas encontradas no ambiente. "A W3C dispõe este documento para guiar a comunidade fornecendo um modelo conceitual junto com um contexto para compreensão dos serviços oferecidos pelos *Web Services* e os relacionamentos destes serviços" [69].

O *WSA* não tem objetivo de especificar como os *Web Services* devem ser implementados ou impor restrições de como os *Web Services* devem ser combinados, mas sim, descrever o mínimo de características comuns para todos os *Web Services*, assim como características que são requisitadas por muitas, mas não por todas as implementações. "Esta documentação tem como público alvo autores de especificações de *Web Services*, desenvolvedores que querem aprofundar seu conhecimento e pessoas que tomam decisões sobre esta tecnologia" [69].

A documentação é organizada em duas sessões principais:

- ***Concepts and Relationships***

Provê um modelo conceitual o qual é baseado em um identificador, por exemplo, um conceito de recurso define que os recursos são identificados nas/pelas URIs *(Uniform Resource Identifier)*. Então, usando esta afirmação como base, avalia-se um recurso particular olhando para o seu identificador. Se em um exemplo dado não tiver nenhum identificador, ele não faz parte desta arquitetura;

- ***Stakeholder's Perspective***

O objetivo desta sessão é fornecer uma vista *top-down* da arquitetura de várias perspectivas. Por exemplo, na sessão de segurança dos *Web Services*, mostra como os serviços de segurança são dirigidos dentro da arquitetura. Com isto, pretende-se mostrar que os serviços de *Web Services* podem ser seguros, indicando os conceitos e características chaves da arquitetura que conseguem este objetivo.

2.4.2 Tecnologias de Apoio aos Web Services

Para entender melhor como as tecnologias trabalham no ambiente *Web Services*, deve-se observar onde estas tecnologias se encaixam no *WSA*. O objetivo destas tecnologias

é definir uma perspectiva de ferramentas que podem ser usadas para projeto, construção e implantação de *Web Services*.

As principais tecnologias dos *Web Services* são o XML [74], SOAP [24], WSDL [11] e UDDI [42].

No entanto, há ainda, uma gama de tecnologias que estendem a capacidade destas ou servem de apoio a elas. Estas outras tecnologias podem ser encontradas no documento "*List of Web Services Specifications Compiled by Roger Cutler and Paul Denning"* [14].

XML

XML (*eXtensible Markup Language*) é uma linguagem muito parecida com HTML *(Hyper Text Markup Language)*, mas com a possibilidade de definir *tags*, que são marcações de dados (*meta–markup language*) [74]. Essas *tags* fornecem um formato para descrever dados estruturados. O XML provê um sistema para criar dados diferentes do HTML que define apenas formatação de textos e caracteres. A Figura 2.6 apresenta um trecho de um arquivo XML.

```
<?xml version="1.0"?>
<!DOCTYPE employees SYSTEM "employees.dtd">
<employees>
        <employee>
                <firstName>Jimi</firstName>
                <lastName>Hendrix</lastName>
                <age>45</age>
                <sex>Male</sex>
                <title>IT Manager II</title>
                <manager>Ace Frehley</manager>
                <phone>801-555-1212</phone>
        </employee>
</employees>
```

Figura 2.6: Trecho de um arquivo XML

XML é a tecnologia chave para atender à comunicação independente nos *Web Services*. As empresas descobriram os benefícios de usar XML para a integração dos dados internamente, compartilhando seus dados entre departamentos, e externamente, compartilhando-os com outras empresas. Graças a isto, XML tornou-se rapidamente o padrão como tecnologia para representação de dados para negócios de Computação na *Web* [74].

SOAP

SOAP (*Simple Object Access Protocol*) é um protocolo que *define* uma estrutura padrão de interoperabilidade, um mecanismo de RPC *(Remote Procedure Call)*, um *framework* para empacotar e desempacotar mensagens XML entre o Servidor e o Cliente [24].

Normalmente, os envelopes, nome dado às mensagens SOAP, são transmitidos via HTTP [27] ou SMTP [52] devido ao seu fácil acesso, mas é possível transmitir SOAP sob praticamente qualquer protocolo.

Uma mensagem SOAP está representada na Figura 2.7. O envelope SOAP (SOAPEnvelope) é a parte obrigatória de uma mensagem SOAP. Ele funciona como um recipiente de todos os outros elementos da mensagem, possivelmente o cabeçalho e o corpo, assim como os *namespaces* de cada um. Um envelope SOAP consiste basicamente em:

- Um cabeçalho composto por zero ou mais entradas;
- Um corpo composto por zero ou mais entradas;

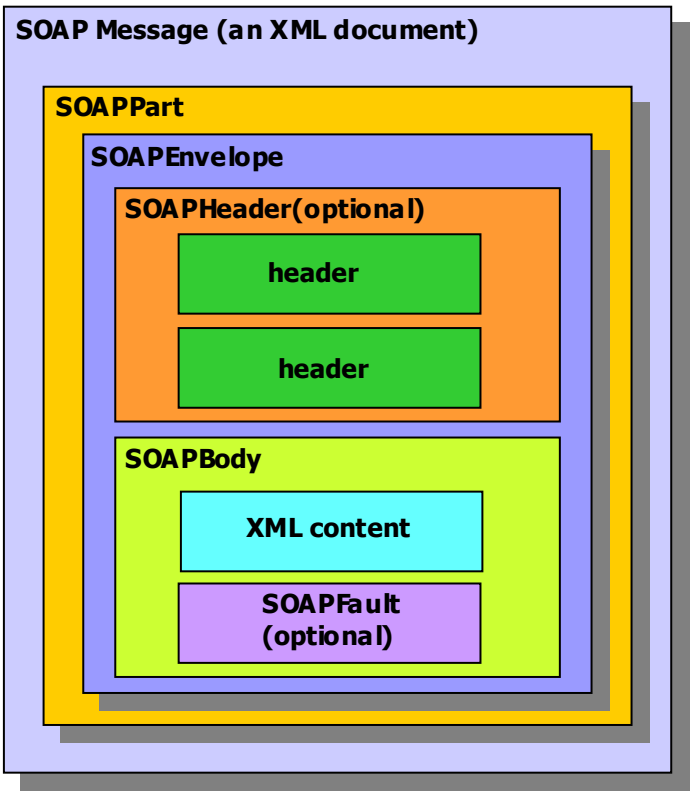

Figura 2.7: Mensagem SOAP

Atualmente, o termo SOAP não mais define um único significado. Existem duas definições para este termo que refletem diferentes caminhos em que a tecnologia pode ser interpretada como:

- *Service Oriented Architecture Protocol:* no caso geral, uma mensagem SOAP representa a informação que você precisa para invocar um serviço ou analisar o resultado desta chamada, junto com a informação específica da definição da interface do serviço;
- *Simple Object Access Protocol:* quando você usa uma representação opcional do SOAP RPC, a mensagem SOAP representa um método de invocação do objeto remoto, e a serialização dos argumentos do método que precisam ser movidos do ambiente local para um ambiente remoto.

WSDL

WSDL (Web Service Description Language) descreve a interface do serviço de forma estruturada e padronizada em XML [11].

O WSDL [11] é a linguagem para descrição de *Web Services*.

Com WSDL especifica-se:

- o que o serviço faz;
- como chamar suas operações;
- onde encontrar o serviço.

WSDL descreve os *Web Services* começando com as mensagens que são trocadas entre os agentes requisitores e provedores. As mensagens são descritas de forma abstrata e limitadas a um formato concreto do protocolo e da mensagem de rede. A Figura 2.8 apresenta a estrutura de um arquivo WSDL.

Os componentes são:

- *wsdl:types*: Tipos de dados;
- *wsdl:message*: Parâmetros de entrada e saída de um serviço;
- *wsdl:operation*: Representa a assinatura do método (operações) e o relacionamento entre parâmetros de entrada e saída;
- *wsdl:portType*: Agrupamento lógico de operações, denominado de tipo de porta;
- *wsdl:binding*: Define o protocolo a ser usado para acessar os métodos de um objeto (SOAP, HTTP ou MIME);

- *wsdl:service*: Endereço do serviço. Além dos componentes acima, esta *tag* define um serviço.

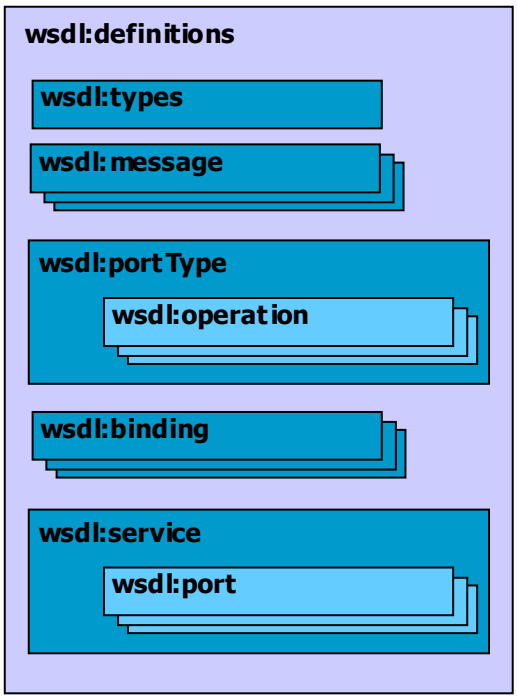

Figura 2.8: Estrutura do Arquivo WSDL

Uma visão mais detalhada do WSDL em uso pode ser definida nas seguintes etapas:

1. O provedor do serviço cria um *Web Service* e registra-o em um *Universal Description, Discovery and Integration* (UDDI). É através do UDDI [42] que um sócio de negócio ou outro sistema ou clientes podem encontrar o serviço;
2. Um requisitor de serviço encontra o diretório de *Web Services* e procura o serviço que lhe interessa. A informação do registro de UDDI é usada para encontrar o documento original WSDL, detalhando a estrutura das chamadas para o(s) serviço(s) desejado(s), que inclui informações tais como parâmetros do serviço e protocolo;
3. Usando a informação fornecida no documento correspondente do WSDL, o programador pode escrever um acesso para o cliente deste serviço, ou simplesmente usar a informação para criar uma mensagem SOAP em baixo-nível usando o protocolo SOAP [24].

UDDI

UDDI (*Universal Description, Discovery and Integration*) corresponde a um *Web Service registry*, que provê um mecanismo para busca e publicação de *Web Services* [42].

Um UDDI *registry* contém informações categorizadas sobre os serviços e as funcionalidades que eles oferecem, e permite a associação desses serviços com suas informações técnicas, geralmente definidas usando WSDL. Como dito anteriormente, o arquivo de descrição em WSDL descreve as funcionalidades do *Web Service*, a forma de comunicação e sua localização. Devido ao modo de acesso, um *UDDI registry* também pode ser entendido como um *Web Service*. A Figura 2.9 apresenta as interações com o UDDI *registry*.

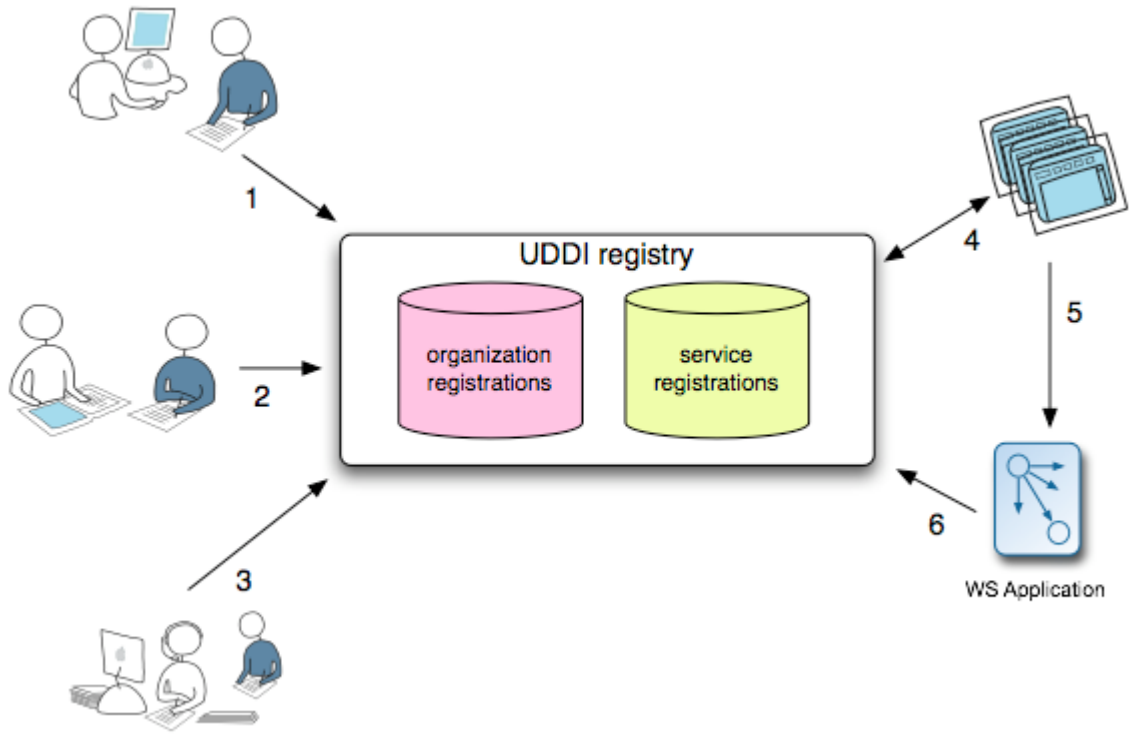

Figura 2.9: Interações com UDDI *registry* [42]

A Figura 2.9 pode ser descrita da seguinte maneira:

1. Os Arquitetos de Negócios populam o *registry* com especificações abstratas dos serviços disponibilizados pela organização. Estas informações são denominadas de modelos técnicos de negócios, ou mais comumente chamados de *tModels*. Em muitos casos, o *tModel* contém a localização do arquivo WSDL que descreve a interface SOAP do *Web Service*;
2. Usuários especializados da organização populam o *registry* com informações sobre a organização e suas unidades;
3. Desenvolvedores, populam o *registry* com informações técnicas dos serviços e os classificam de acordo com o *tModel* definido pelos Arquitetos de Negócios;

4. Consumidores de serviços consultam o *registry* procurando por serviços que atendam as suas necessidades;
5. Os serviços descobertos são ligados para a criação de novas aplicações, podendo gerar novos *web services* ou não;
6. Aplicações podem dinamicamente consultar o *registry* para localizar *end points* de serviços ou outros metadados.

A especificação UDDI define uma API baseada em mensagens SOAP, com uma descrição em WSDL do próprio *Web Service* do servidor de registro. A maioria dos servidores de registro UDDI também provê uma interface de navegação por *browser*.

2.4.3 Funcionamento

De outra forma, pode-se descrever que os *Web Services* são artefatos de softwares acessados por aplicações remotas. Para isto, as mensagens trocadas são formatadas no protocolo SOAP, o que garante a interoperabilidade entre diferentes plataformas, em um processo denominado serialização XML, permitindo que classes sejam mapeadas para esquemas XML e que instâncias de objetos sejam mapeadas para instâncias XML do seu esquema correspondente. Porém, antes que as mensagens SOAP sejam trocadas, suas características são explicitadas através de documentos WSDL, que descrevem quais dados estarão sendo trocados, e como estes dados estarão organizados nas mensagens SOAP.

2.4.4 *Web Services Protocols* (WS-*)

Grande parte da promessa dos *Web Services* é permitir a comunicação entre sistemas com diferentes sistemas operacionais e plataformas de desenvolvimento. Para tornar isso possível, os serviços são baseados em uma família de especificações de protocolos da indústria para os *Web Services*, geralmente denominada WS-*. Aspectos como segurança, garantia de entrega de mensagens, uso de transações, encaminhamento e coordenação de *Web Services* são alguns pontos que estão sendo discutidos para tornar o uso de *Web Services* cada vez mais padronizado. Com este objetivo, muitas propostas estão sendo submetidas aos órgãos como W3C (*World Wide Web Consortium*) e OASIS (*Organization for Structured Information Standards*).

WS-Security

Padronizada pela OASIS, a *WS-Security* [43] descreve melhorias e extensões nas mensagens SOAP para agregar as propriedades de confidencialidade e integridade. É possível utilizar uma grande variedade de mecanismos de segurança e tecnologias para cifragem, como por exemplo, infra-estrutura de chave pública (ICP), Kerberos [38] ou SSL [28]. Essa especificação foi projetada para ser expansível, permitindo o uso de múltiplas credenciais, possibilitando ao cliente utilizar diferentes formatos de credenciais para a autenticação e autorização.

WS-Policy

A especificação WS-Policy [70] provê uma gramática flexível que possibilita expressar políticas em sistemas baseados em XML. A especificação inclui um conjunto de asserções gerais, relacionadas às mensagens SOAP; estas asserções são definidas pela especificação *WS-PolicyAssertion* [70]. Asserções relacionadas à segurança, que suportam a especificação *WS-Security*, são tratadas na especificação *WS-SecurityPolicy* [70]. A definição de qual algoritmo usar para cifrar e assinar as requisições a um determinado *Web Services* seria uma das habilidades descritas pela *WS-SecurityPolicy*.

A *WS-Policy* não descreve como essas políticas são divulgadas ou como anexá-las a um *Web Service*, assunto coberto pela especificação *WS-PolicyAttachment* [70], que define como anexar as políticas com elementos XML, WSDL e UDDI.

WS-Addressing

Já a *WS-Addressing* [71], desenvolvida pela IBM, Microsoft e Bea, provê uma estrutura para fazer o endereçamento e encaminhamento de mensagens de serviços de forma independente do transporte. Esta especificação também dispõe de uma série de informações sobre cabeçalhos usados comumente nos textos. Combinado ao *WS-ReliableMessaging* [44], permite transporte neutro e interações bidirecionais entre *Web Services* por meio de rede, que incluem gerenciadores de *end-point*, *firewalls* e *gateways*.

O *WS-Addressing* é um componente-chave para a obtenção da segurança no nível da mensagem, pois fornece os mecanismos necessários para lidar com as mensagens de uma forma independente do transporte. Isso permite que uma mensagem segura seja enviada através de qualquer transporte e seja facilmente roteada.

WS-ReliableMessaging

A *WS-ReliableMessaging* [44] em conjunto com *WS-Addressing* estão destinadas a garantir segurança e entrega de mensagens via a Internet. Esta especificação provê um protocolo capaz de detectar se uma mensagem não foi recebida ou duplicada em meio às transações de *Web services*. Além disso, oferece recursos para que as mensagens recebidas sejam processadas na ordem em que foram enviadas.

WS-Coordination

A especificação *WS-Coordination* [72] descreve um *framework* extensível para prover protocolos que coordenem as ações de aplicações distribuídas. O *fremework* definido nesta especificação permite a uma aplicação ou um serviço criar o contexto necessário para propagar as suas atividades para outros serviços e para registrar protocolos para coordenação. O *framework* permite a existência de processamento de transações, *workflow* e outros sistemas para coordenação ocultar seus protocolos proprietários e operar em um ambiente heterogêneo. *WS-Coordination* é uma especificação que tenta padronizar uma infra-estrutura de conversação entre clientes e servidores e também padronizar a coordenação das atividades distribuídas desempenhadas por clientes e servidores no ato de transações realizadas entre diferentes sistemas [72].

2.4.5 WS-I

A *Web Services Interoperability Organization* (WS-I) [73] é um consórcio formado para disponibilizar e promover a interoperabilidade entre as soluções baseadas em *Web Services*. A WS-I define perfis para conjuntos de aplicações, oferece auxílio para as implementações e aplicações de demonstração, ferramentas e materiais para a execução de testes para os desenvolvedores de sistemas baseados em *Web Services* para que sejam praticadas regras que assegurem a interoperabilidade.

O perfil básico regulamentado exige suporte aos padrões WSDL 1.1, SOAP 1.1, HTTP 1.1, SSLv3 e UDDI 2.0. O trabalho da WS-I está influenciando fortemente como a arquitetura dos *Web Services* será estabelecida nos próximos anos.

O problema da Interoperabilidade vai para além da representação de dados e concentra-se na compatibilidade das implementações das normas. Os pontos de partida para a interoperabilidade são as normas de todas as categorias, que por vezes não têm

especificações suficientemente claras, o que obriga a ter mecanismos de verificação da compatibilidade.

As normas são as seguintes:

- **Normas de Transporte:** Resolvem o problema de estabelecer um canal de comunicação entre o cliente e o prestador do serviço. A comunicação pode ser síncrona ou assíncrona, e adicionalmente pode oferecer garantias de confiabilidade ou segurança;
- **Normas de Mensagem:** Definem a estrutura das unidades de comunicação e as formas como são trocadas entre os serviços. O SOAP define mensagens em XML, separando o cabeçalho com dados extensíveis para a plataforma, do corpo com dados para as aplicações;
- **Normas de Contrato:** Resolvem o problema da descrição da interface, da política e dos recursos do serviço. A interface do serviço é especificada de forma rigorosa com a WSDL 1.1. A política do serviço, ou seja, os requisitos que têm que ser cumpridos pelo cliente e pelo prestador do serviço para que a interação entre ambos possa acontecer;
- **Normas de Descoberta:** Definem formas de publicar e pesquisar serviços. A UDDI 1.2 define um diretório de pesquisa;
- **Normas de Transações:** Permitem ter semânticas bem definidas para os resultados de várias interações entre serviços com recursos informacionais distribuídos[13]. Para isso, assumem modelos de faltas temporárias e recuperáveis para as máquinas e comunicações;
- **Normas de Segurança:** Especificam mecanismos de proteção para que os *Web Services* possam ser utilizados em aplicações com valor. As operações de base permitem a troca de itens de segurança e formas de garantir a integridade, a autenticação e a confidencialidade de mensagens;
- **Normas de Processos de Negócio:** Satisfazem a necessidade de ferramentas cujo nível de abstração de conceitos é menos técnico e mais próximo do negócio e das preocupações das pessoas da organização.

Assim, a WS-I [73] é a organização que junta os principais fornecedores de ferramentas para definir perfis de interoperabilidade. Cada perfil abrange um conjunto de

[13] Recursos informacionais distribuídos são recursos de dados utilizados por mais de um serviço, acessado e utilizado através de transações concorrentes.

normas e fornecem orientações à sua implementação, aplicações de exemplo e testes para aferir o cumprimento das especificações.

2.5 MDA

Em 2000, a OMG (*Object Management Group*), inspirada na importância dos modelos em processos de desenvolvimento de software e na sua missão de prover soluções para o problema da interoperabilidade entre os sistemas, criou o *framework* MDA (*Model-Driven Architecture*) [45]. O MDA [45] utiliza modelos formais, ou seja, que possam ser entendidos e interpretados por computadores. Esse *framework* define e especifica alguns modelos, como eles devem ser usados e o relacionamento entre eles durante todas as fases do ciclo de vida de um sistema. As principais promessas do MDA são a portabilidade, interoperabilidade e reusabilidade e baseia-se em 3 padrões criados pela OMG: UML (*Unified Modeling Language*) [46], MOF (*Meta Object Facility*) [47] e CWM (*Common Warehouse Metamodel*) [48].

O MDA traz consigo:

- Benefício da portabilidade e da eficiência, preservando o investimento nas tecnologias através da modelagem independente de plataformas;
- Velocidade no desenvolvimento através da geração automática de código;
- Qualidade da implementação através do desenvolvimento de *templates* por *experts* ;
- Facilidade de manutenção e documentação visto que o projeto e os modelos não são abandonados após a escrita.

O MDA não se aplica somente ao tratamento de modelos, como as regras de transformação para a geração de PSMs (*Plataform Specific Model*) a partir de um PIM (*Plataform Independent Model*). Este também é utilizado no processo de gerenciamento de metadados. Quando MDA é utilizado para o gerenciamento de metadados, os padrões MOF (*Meta Object Facility*) [47], XMI (*XML Metadata Interchange*) [49] e JMI (*Java Metadata Interface*) [30] são de elevada importância.

2.5.1 MOF

Meta-Object Facility (MOF) [47] é um padrão da OMG que especifica uma linguagem abstrata para descrever outras linguagens. Neste contexto, linguagem significa

uma sintaxe abstrata de uma linguagem, isto é, MOF não é usado para descrever uma gramática, ele é usado para descrever a estrutura dos objetos que podem ser representados em uma dada linguagem. O MOF é também freqüentemente referenciado como um meta-metamodelo e as sintaxes abstratas usadas para descrevê-los são chamados de metamodelos.

MOF é o mecanismo básico do MDA para definição de linguagens de metamodelos. Ele provê um conjunto de conceitos para defini-los, em particular: Diagramas de Classe para definir a sintaxe abstrata e OCL para definir semânticas de uma linguagem de modelagem. Como exemplo, temos o metamodelo da UML, onde a semântica é definida usando uma mistura de OCL e texto informal.

A Figura 2.10 apresenta um trecho de um metamodelo da XML. Neste fragmento, os elementos que fazem parte de um documento XML e a relação entre estes são apresentados. Neste fragmento de Metamodelo XML representado na Figura 2.10, temos que um documento XML é composto por nós. Esses nós podem ser elementos, atributos, textos, entre outros. Elementos podem conter atributos, textos ou outros elementos aninhados. Um elemento contém outros nós através de uma agregação. Cada documento XML possui um nó raiz do qual todos os outros nós do documento XML são filhos.

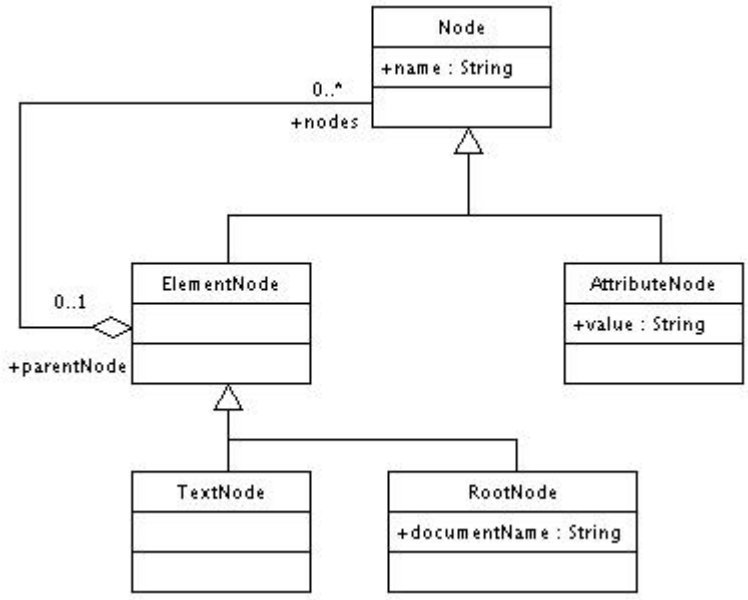

Figura 2.10: Metamodelo XML [47]

Desta forma, o MOF é o caminho padrão para definir linguagens de metamodelagens. No entanto, este não é apenas aplicado a OO, isto é, este pode ser usado para definir linguagens de metamodelagens não orientadas a Objeto.

Outra característica do MOF é fornecer a serialização de modelos e APIs para manipulação livre dos modelos. Seu escopo não se restringe apenas para a criação de linguagens de metamodelagens, mas também para o gerenciamento de metadados.

Neste objetivo de manipulação de metadados duas tecnologias padrões são de grande importância e trabalham em conjunto com o MOF:

- XMI (*XML Metadata Interchange*) que provê a serialização de um modelo (XML *mapping*);
- JMI (*Java Metadata Interface*) que provê APIs para manipulação de modelos (Java *mappings*).

2.5.2 XMI

O XMI é um padrão de troca de modelos e metadados utilizado entre várias ferramentas, repositórios e *middlewares* que necessitam de um alto grau de interoperabilidade. O XMI é o caminho padrão para mapeamento de objetos em XML (XML não é orientado a objeto).

Na Figura 2.11, o trecho de um arquivo XMI é apresentado. Neste trecho, podemos perceber a existência de uma *tag Model:Class* que representa uma Classe e suas informações, por exemplo, seu nome, se a classe é abstrata, sua visibilidade dentre outras informações. Outra *tag* disponível neste trecho de arquivo é *Model:Attribute* que representa um Atributo de uma Classe. Esta *tag* também tráz informações sobre o atributo tais como, seu nome, seu escopo, sua visibilidade dentre outras informações.

```
<?xml version = '1.0' encoding = 'windows-1252' ?>
<XMI xmi.version = '1.2' xmlns:Model = 'org.omg.xmi.namespace.Model' timestamp = 'Sun Jul 16 15:59:20 GMT-
03:00 2006'>
  <XMI.header>
    . . .
  </XMI.header>
  <XMI.content>
    <Model:Class xmi.id = 'a52' name = 'ProxyAgent' annotation = '' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
          visibility = 'public_vis' isSingleton = 'false'>
        <Model:Attribute xmi.id = 'a59' name = 'name' annotation = '' scope = 'instance_level' visibility = 'public_vis'
          isChangeable = 'true' isDerived = 'false'>
        </Model:Attribute>
      </Model:Class>
      <Model:Import xmi.id = 'a61' name = 'PrimitiveTypes' annotation = '' visibility = 'public_vis'
        isClustered = 'false'><Model:Import.importedNamespace><Model:Package xmi.idref = 'a1'/>
</Model:Import.importedNamespace> </Model:Import></Model:Namespace.contents> </Model:Package>
  </XMI.content>
</XMI>
```

Figura 2.11: Trecho de um arquivo XMI [49]

O XMI define dois conjuntos de regras:

- Um conjunto para serialização de objetos para documentos XMIs;
- Outro conjunto para geração de esquemas XMLs dos modelos.

Versões antigas do XMI definiam um conjunto de regras para geração de DTDs, atualmente este utiliza-se do XML *Schema*.

O principal uso do XMI é a serialização de objetos de nível Meta, tais como dados, metadados e meta-metadados.

Na escrita de objetos usando XMI teremos:

- Cada objeto mapeia para um elemento XML;
- A identidade de um objeto é implementada usando atributos XML;
- Atributos de Dados mapeiam para atributos XML ou são aninhados em elementos XML (de acordo com a necessidade de múltiplos ou nenhum elemento);
- Atributos de Objetos mapeiam para elementos XMLs aninhados (nomes de atributos de objetos transformam-se em nome de elementos XML);
- Referências de Composições de objetos mapeiam como atributos de objetos;
- Referências (Instâncias e Associações) mapeiam para atributos XMLs;
- Informações adicionais em um elemento mapeia para a *tag Extension* em XML.

Na geração de Esquemas de Modelos, as regras são mais complexas que as de serialização de objetos.

Os conceitos de modelos usados no mapeamento para Esquemas incluem:

- Pacotes;
- Classes;
- Tipos de Dados;
- Atributos;
- Associações;
- Herança.

No mapeamento, informações são perdidas, tais como tipos de atributos de objetos.

Os metamodelos podem ser serializados conforme o MOF XMI ou UML XMI usando um MOF profile. Como exemplo, temos a ferramenta uml2mof que acompanha o MDR (um repositório MOF do NetBeans) [37].

No entanto, o XMI também apresenta algumas dificuldades tais como:

- Não é diretamente entendível para os humanos;
- Problema com a evolução dos modelos (a cada menor mudança realizada num modelo automaticamente transforma o documento em um XMI inválido);

2.5.3 JMI

A especificação JMI [30] permite a implementação de uma infra-estrutura dinâmica independente de plataforma para gerenciar a criação, o armazenamento, acesso, descoberta e troca de metadados conforme a MOF.

JMI é um padrão fornecido pela JCP (Java *Comunity Process*), baseado na especificação MOF da OMG. Assim, JMI é um padrão endossado pela indústria para gerenciamento de metadados. Modelos e qualquer tipo de metadados (chamado de metamodelos) podem ser construídos a partir destes blocos básicos de construção. O JMI define interfaces Java padrão para estes componentes de modelagem, e através destas permite a descoberta e o acesso aos metadados independente de plataforma. JMI permite a descoberta, a consulta, o acesso e a manipulação de metadados em tempo de projeto e em tempo de execução. A semântica de qualquer sistema modelado pode ser completamente descoberta e manipulada. O JMI também permite a interação entre modelos e metadados através da especificação XMI.

Existem muitas implementações do JMI, incluindo a *Reference Implementation* da Unisys [66], a implementação *open-source* do grupo NetBeans.org [37] e muitas outras implementações que são parte de vários produtos industriais.

2.5.4 MDR

O MDR (*Metadata Repository*) [37] implementa um repositório de metadados baseado no padrão MOF da OMG e está sendo desenvolvido como parte do projeto NetBeans. Este inclui a implementação de um repositório MOF com um mecanismo de armazenamento persistente para o armazenamento de metadados. A interface do repositório MOF é baseada no JMI. O MDR também define características adicionais que ajudam a incorporá-lo em um IDE. Assim, visto que este implementa MOF, está habilitada a carregar qualquer metamodelo MOF (descrição de metadados) e armazenar instâncias deste metamodelo (metadados conforme o metamodelo). Metamodelos e metadados podem ser importados ou exportados de/para o MDR usando o padrão XMI. Os metadados no repositório podem ser gerenciados através de rotinas usando API JMI. A Figura 2.12

apresenta a estrutura do projeto MDR. Nesta Figura são apresentados os módulos e o relacionamento de associação entre estes no processo de composição do *MDR Module*. Na Figura 2.12 os retângulos maiores, representam os módulos NetBeans que compõem o MDR. Cada módulo MDR NetBeans consiste de um ou mais arquivos *jar* ou bibliotecas externas (representadas na figura pela palavra ext). Os retângulos menores representam as bibliotecas (arquivos *jars*). As setas de dependências representam a dependência entre os módulos. O módulo NetBeans origem da seta, representa o módulo que depende do módulo alvo da seta.

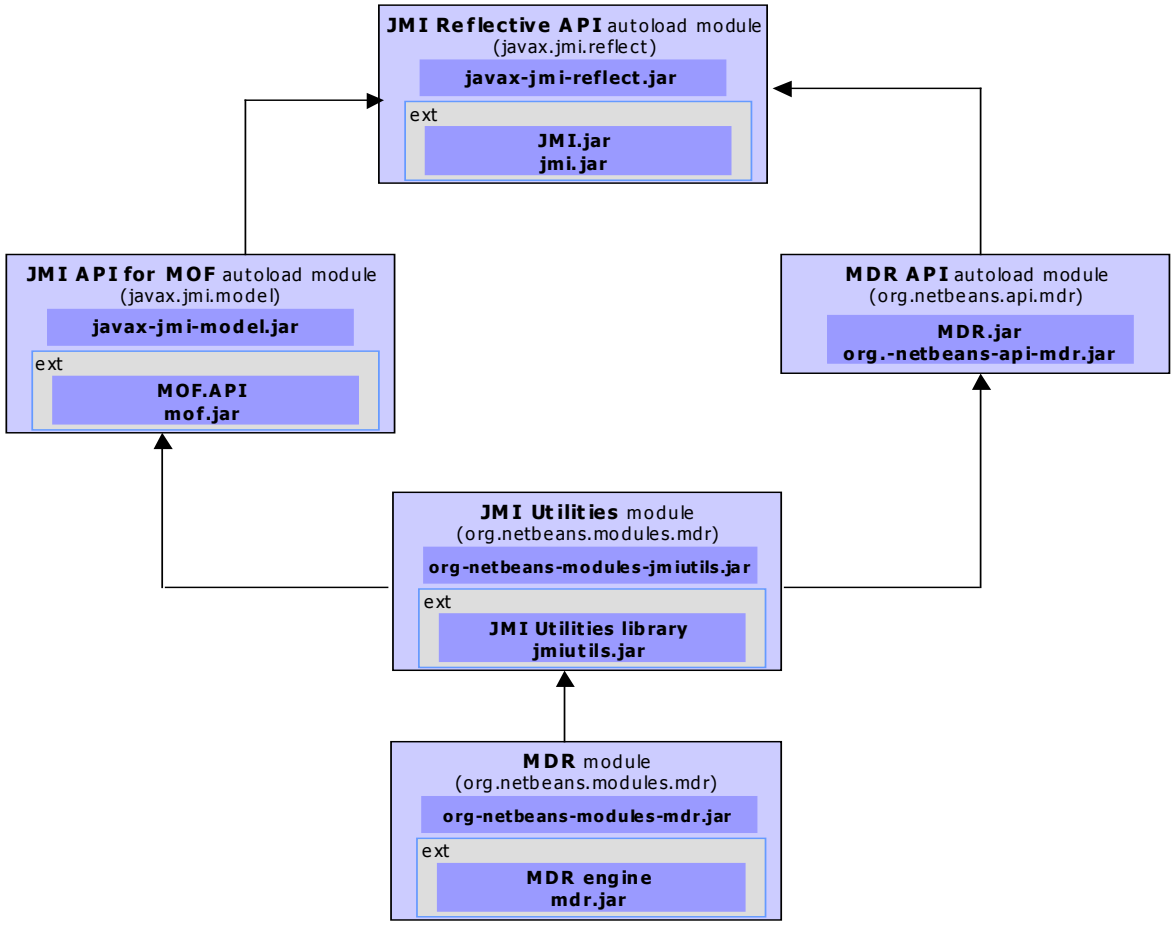

Figura 2.12: Estrutura do Projeto MDR

Para conhecer todas as funcionalidades do MDR é necessário que o usuário escreva um módulo que faça uso do MDR.

As características básicas do MDR podem ser vistas a seguir:

- Suporte a importação de documentos XMI 1.1/1.2 para ambos os formatos MOF 1.4 e MOF 1.3. Quando usado o MOF 1.3 este é transparentemente convertido para MOF 1.4;

- MDR está habilitado a carregar qualquer metamodelo MOF 1.4 (e mesmo MOF 1.3) no formato de um documento XMI. Os metamodelos MOF 1.3 são automaticamente convertidos a metamodelos MOF 1.4 durante o processo de importação;
- Suporte a exportação de dados armazenados em um repositório para documentos XMI 1.2;
- O MDR está habilitado a salvar o conteúdo de qualquer pacote extendido em um documento XMI 1.2;
- Geração de interfaces Java (conforme JMI) para qualquer metamodelo MOF carregado;
- MDR está habilitado a gerar APIs Java para acessar metadados descritos pelo metamodelo MOF;
- Instanciação de qualquer metamodelo de acordo com MOF. Qualquer metamodelo MOF carregado no MDR pode ser instanciado (isto pode ser entendido como a criação de um sub-repositório habilitado a armazenar metadados descritos neste modelo). A instância do modelo pode então ser acessado usando APIs reflectivas ou APIs geradas especificamente para o metamodelo.

2.6 Conclusões

Neste capítulo, alguns conceitos relacionados aos sistemas de detecção de intrusão, aos sistemas multiagentes, aos *Web Services* e a MDA foram apresentados.

Na seção de sistemas de detecção de intrusão, as características, a classificação e as limitações desse tipo de sistema foram apresentadas. Na seção de sistemas multiagentes, a definição de agente, sua classificação, forma de interação e plataformas existentes foram abordadas. Na seção de projetos de sistemas de detecção baseados em agentes, alguns projetos e suas principais características foram apresentados. Na seção de *Web Services*, as tecnologias de apoio, seu funcionamento e a especificação dos seus principais protocolos de utilização foram apresentados. Na seção de MDA, as principais tecnologias de apoio que MDA utiliza foram apresentadas.

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

constatar que a maioria dos projetos tem o objetivo de proteger a rede local em que está situado, não citando a possibilidade de usar o poder da ferramenta para proteger usuários externos.

3 Projeto NIDIA

A proposta do sistema NIDIA (*Network Intrusion Detection System based on Intelligent Agents*) [36] é apresentar um sistema de detecção de intrusão, composto por um conjunto de agentes, fornecendo um modelo de detecção de intrusos, em tempo real, baseado na noção de sociedade de agentes inteligentes capaz de detectar novos ataques através de uma rede neural.

O NIDIA é inspirado no modelo lógico do CIDF [60], possuindo para este fim, agentes com função de geradores de eventos (agentes sensores), mecanismos de análise dos dados (agentes de monitoramento e de avaliação de segurança), mecanismos de armazenamento de histórico (bases de dados) e um módulo para realização de contramedidas (agente controlador de ações). Além disso, existem agentes responsáveis pela integridade do sistema e pela coordenação das atividades do IDS como um todo.

Desta forma, os agentes do NIDIA possuem os seguintes objetivos gerais:

- Gerar índices de suspeita de ataque a partir da análise de dados coletados de *logs* de *hosts* e de pacotes de tráfego de rede;
- Tomar contramedidas de acordo com os índices obtidos;
- Aprender com os casos obtidos atualizando suas bases de conhecimento.

O modelo proposto prevê a metodologia de detecção por abuso e anomalia para garantir uma robustez maior ao sistema. Entretanto, atualmente, implementou-se somente a detecção por abuso como método de análise. A escolha se deu em virtude da grande maioria dos ataques poderem ser codificados, de maneira a capturar e registrar variantes das atividades que exploram as mesmas vulnerabilidades.

A escolha da arquitetura multiagentes [25] para o IDS NIDIA visa obter as seguintes vantagens:

- Agentes podem ser adicionados ou removidos para/do sistema sem modificar outros componentes do sistema;
- Agentes podem ser reconfigurados ou atualizados sem causar problemas no restante do sistema;
- Um agente ou um grupo de agentes pode realizar diferentes funções simples. Visto que os agentes podem trocar informações entre si, podem derivar resultados mais complexos.

Através da arquitetura em camadas do NIDIA, o objetivo específico dos agentes que o compõe será demonstrado.

3.1 Arquitetura do NIDIA

A Figura 3.1 mostra a arquitetura do Sistema NIDIA que é composto por camadas [36]. Cada camada possui atividades a desempenhar, sendo que estas atividades são executadas através dos comportamentos dos agentes que a compõe. É através destes agentes também, que as camadas se comunicam trocando informações importantes para desempenhar as suas atividades.

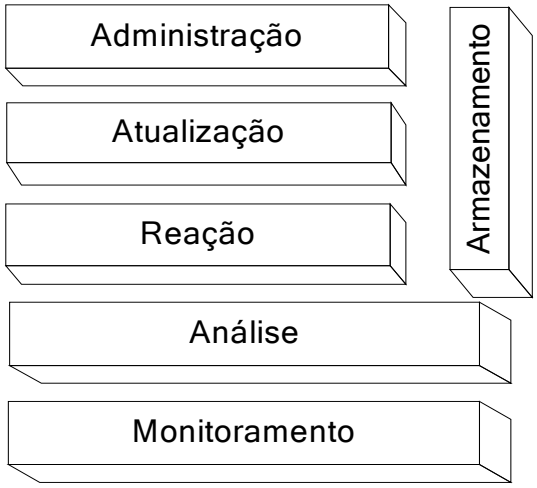

Figura 3.1: Modelo em Camadas do NIDIA [36]

Segue abaixo um descritivo das funcionalidades das camadas do NIDIA e os respectivos agentes que a compõem:

- **Camada de Monitoramento**

Camada responsável por capturar a ocorrência de eventos no meio exterior e fornecer informações sobre o mesmo para o resto do sistema. Nesta camada, os Agentes Sensores de Rede e Agentes Sensores de *Host* estão localizados. Estes agentes funcionam como os "sentidos receptores" do sistema. Segue um breve descrição destes agentes:

- **Agentes Sensores de Rede**: responsáveis por capturar os pacotes que estão trafegando na rede. Estes atuam em pontos estratégicos e funcionam como monitores de rede passivo, trabalhando em modo promíscuo, desta forma não interferindo na performance e nem no tráfego da rede;
- **Agentes Sensores de *Host***: trabalham coletando informações em tempo real de um *host* em particular (geralmente servidores) e disponibilizando-as para análise;

Os dados obtidos recebem uma pré-formatação sendo em seguida repassados para o agente de avaliação de segurança.

- **Camada de Análise**

Camada responsável pela análise dos eventos recebidos da camada de monitoramento. Nesta camada os eventos coletados são formatados pelo agente SMA (*System Monitoring Agent*) e posteriormente analisados pelo agente SEA (*Security Evaluation Agent*). De maneira que, padrões de ataques possam ser identificados e, posteriormente, confirmados como verdadeiro ataque. Para isso utiliza bases de conhecimento, como a base de dados de padrões de intrusões (IIDB), a base de dados de incidentes de intrusão (DFDB) e a base de estratégias (STDB). Nesta camada, localizam-se os agentes SMA e SEA. Estes são responsáveis por formatar os dados e realizar a análise dos eventos coletados e emitir um grau de suspeita sobre os eventos que foram previamente formatados.

- **Camada de Reação**

Camada responsável por tomar contramedidas caso um incidente de segurança seja detectado. Com base no parecer do SEA, agente da Camada de Análise, a Camada de Reação deve tomar uma contramedida de acordo com as bases de dados de estratégia (STBD) e de ações (RADB). Nesta camada, localizam-se os agentes SCA (*System Controller Agent*).

- **Camada de Atualização**

Camada responsável pela atualização das bases de informações. As consultas poderão ser feitas diretamente de qualquer camada, porém inserções devem ser feitas somente através desta camada. Ela terá também a responsabilidade de manter a integridade e consistência das informações armazenadas. Nesta camada, localizam-se os agentes SUA (*System Updating Agent*). Estes são responsáveis pela atualização das bases DFDB, IIDB, RADB e STDB.

- **Camada de Administração**

Camada responsável pela administração e integridade de todos os agentes do sistema. Nesta camada, localizam-se os agentes MCA (*Main Controller Agent*).

- **Camada de Armazenamento**

Camada responsável por manter de forma persistente informações provenientes das demais camadas. Nesta camada, localizam-se as bases de dados utilizadas pelo NIDIA. Segue uma breve descrição das mesmas:

- STBD (*Strategy DataBase*) é uma base de dados responsável por registrar as estratégias adotadas por uma organização qualquer em relação à sua política de segurança. Ela é importante para garantir a adaptabilidade do IDS aos mais diversos casos;
- RABD (*Reaction DataBase*) estão contidas as informações referentes às ações que devem ser tomadas de acordo com a severidade do ataque detectado. Também varia de acordo com a política de cada instituição;
- IIDB (*Incidents of Intrusion DataBase*) guarda as assinaturas de intrusão que serão utilizadas para a detecção de atividades suspeitas. Ele deve ser constantemente atualizado para garantir que novas técnicas de ataque possam ser detectadas;
- DFDB (*Standard of Intruders and Intrusions DataBase*) registra os danos causados por ataques bem-sucedidos e tentativas de ataques. Nele ficam contidas as informações que podem ser úteis na identificação de tentativas de ataques provenientes de uma mesma origem ou domínio ou simplesmente para serem utilizadas em investigações futuras.

3.2 Funcionamento

Funcionamento do sistema NIDIA pode ser demonstrado através da interação entre os agentes que compõe suas camadas. Cada camada dentro do NIDIA possui um objetivo bem definido. Este objetivo é alcançado através da interação e troca de informações entre as camadas. Para um melhor entendimento, o funcionamento do IDS NIDIA será dividido em três partes: Funcionamento da Captura, Funcionamento da Análise e Funcionamento das Contramedidas. Nas próximas seções, o Diagrama de Seqüência da UML é utilizado para um melhor entendimento do explicitado aqui.

3.2.1 Funcionamento da Captura

Os agentes sensores de *host* e de rede são os responsáveis por capturar a ocorrência de eventos e fornecer informações sobre os mesmos para o resto do sistema. Os agentes sensores de *host* fazem a leitura dos *logs* de segurança de servidores específicos e repassam para o agente SMA de *host* para que o mesmo formate os dados. Posteriormente tais informações são enviadas para o agente SEA de *host* para que seja feita a avaliação dos dados provenientes do SMA.

Os agentes sensores de rede capturam pacotes do tráfego e realizam duas tarefas:

- Enviam o cabeçalho dos pacotes para os agentes SMA de rede específicos para formatar esse tipo de informação;
- Enviam os pacotes contendo os dados sobre as conexões para os agentes SMA de rede específicos para formatar esse tipo de informação. Um das informações enviadas no conteúdo dos pacotes recolhidos, são informações sobre o endereço do emissor e o receptor da mensagem de forma que essa conexão TCP possa ser identificada.

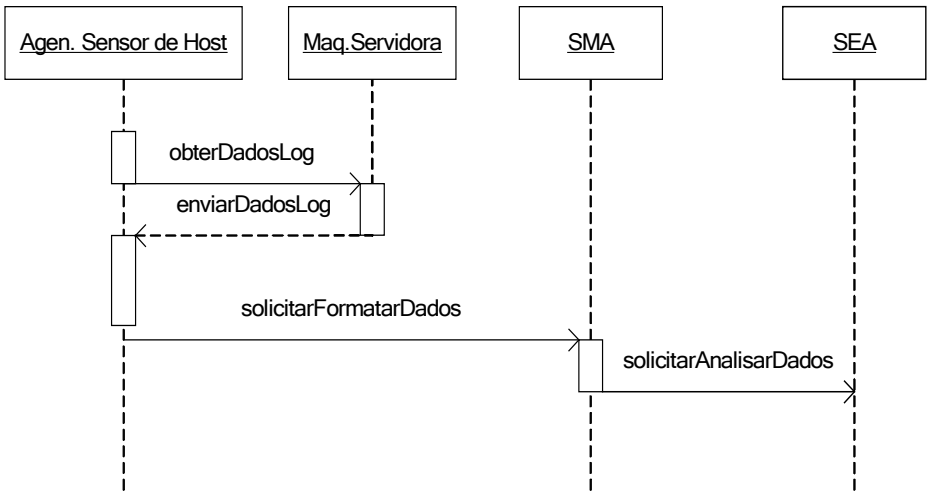

Figura 3.2: Diagrama de Seqüência do Funcionamento da Captura

Os agentes SMA realizam a tarefa de formatar os dados recebidos. Porém, no caso específico do tratamento das conexões TCP remontadas pelo agente sensor de rede, o agente SMA especializado além de formatar, consulta um banco de dados de palavras-chaves para a contagem de ocorrência de palavras-chaves que indiquem um ataque.

Posteriormente, os agentes SMA enviam os dados pré-processados para os agentes SEA avaliarem.

3.2.2 Funcionamento da Análise

A Figura 3.3 mostra o diagrama de seqüência do funcionamento da análise. Existem agentes SEA especializados em tratar o cabeçalho dos pacotes através de filtros, auxiliado pelas bases de informações pertencentes ao NIDIA. E existem agentes SEA especializados em inspecionar o conteúdo das conexões TCP. O produto gerado pelos agentes SEA é o grau de suspeita dos dados verificados que é enviado para o Agente Controlador de Ações (SCA), para caso necessário, tomar alguma contramedida baseada em um banco de ações.

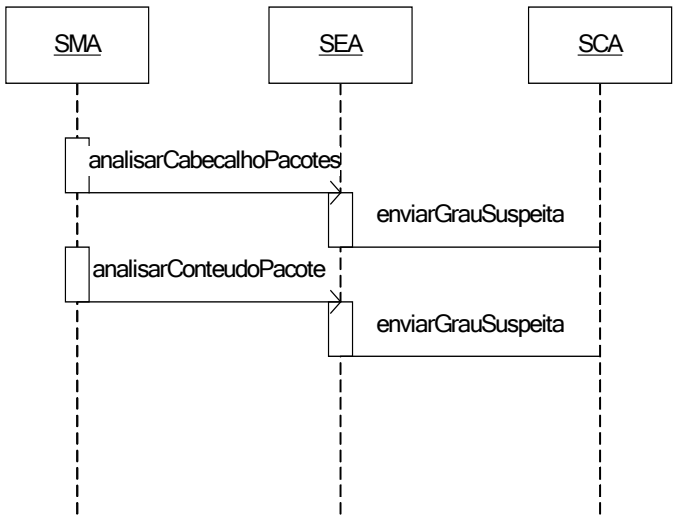

Figura 3.3: Diagrama de Seqüência do Funcionamento da Análise

3.2.3 Funcionamento da Contramedida

A Figura 3.4 apresenta o diagrama de seqüência do funcionamento da contramedida. O produto gerado pelos agentes SEA é o grau de suspeita dos dados verificados que é enviado para o Agente Controlador de Ações (SCA), para caso necessário, tomar alguma contramedida baseada no STBD. Quando de um ataque não registrado ainda nas bases do NIDIA, o SCA solicita que as atualizações sejam feitas, a fim de registrar as informações sobre o ataque no DFDB. Isto é importante para que se tenham dados das origens das tentativas de intrusão e intrusões consumadas, bem como os danos causados, históricos e notificação dos incidentes de segurança.

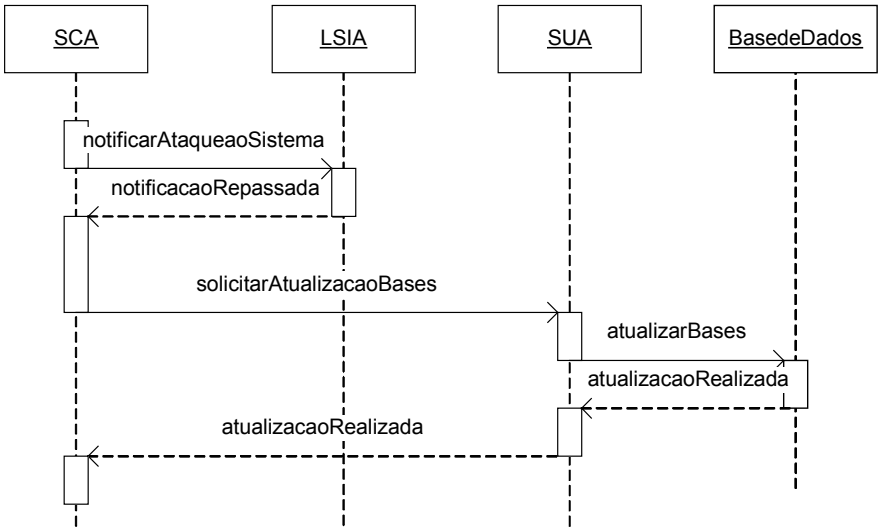

Figura 3.4: Diagrama de Seqüência do Funcionamento da Contramedida

Por fim, tem-se o LSIA (*Local Security Intelligent Agent*) que é responsável pelo controle dos demais agentes, verificando o estado do IDS como um todo e por servir de interface ao agente humano. Se algo estiver errado, a equipe de segurança deve ser informada. Aliás, esse é o único agente que se comunica diretamente com o administrador do sistema, realizando tarefas como identificar quais agentes estão ativos, onde estes estão executando, quais as configurações de cada um, etc.

Outro agente que se comunica com todos os outros agentes da sociedade é o Agente de Integridade do Sistema (SIA). Sua função é verificar se o próprio IDS foi alvo de ataque. Para isso, o perfil de cada agente é definido com os respectivos limites de normalidade. Se essa barreira for ultrapassada por algum agente, é bastante provável que o sistema esteja corrompido e, portanto, medidas devem ser tomadas assim como informar o ocorrido ao administrador.

Convém ressaltar que, como em qualquer sistema multiagentes, os eventos do modelo não ocorrem de forma seqüencial, pois cada agente é executado em uma linha de execução independente e de forma distribuída (em vários servidores e estações de trabalho), competindo e cooperando entre si para realizarem tarefas, de forma a proteger o ambiente computacional sem, preferencialmente, gerar impacto negativo no desempenho da rede.

3.3 Inovações dentro do IDS NIDIA

Durante o ciclo de desenvolvimento do projeto do IDS NIDIA, várias inovações foram inseridas em seu modelo. Devido ao constante crescimento e variedade dos ataques, surgiu a necessidade de que novas funcionalidades fossem embutidas no projeto para tornar o sistema mais robusto e capaz.

Tais funcionalidades geraram atualizações aos agentes que pertencem às camadas do NIDIA. Estas inovações podem ser mais bem entendidas através dos trabalhos [17], [18], [51] e [55].

A primeira inovação aplicada ao IDS NIDIA foi a criação de um conjunto de agentes responsáveis pelo processo de análise dos pacotes disponibilizados após a formatação. Isto torna o processo de análise bem mais confiável. Tal objetivo foi alcançado através de uma sociedade de agentes denominados SAARA (Sociedade Atualizada de Agentes de Reconhecimento de Assinaturas) [17].

A segunda inovação foi à criação de agentes que tornassem o índice de resposta do IDS bem mais eficiente, criando desta forma, uma sociedade de agentes inteligentes que fornece respostas preventivas e ofensivas ao sistema NIDIA. Surgiram então agentes responsáveis pelo gerenciamento do controle central (MCA - *Main Controler Agent*), pela avaliação da segurança (BAM - *Binary Association Memory*) , agentes de sistema operacional (SOA - *System Operational Agent*) e agentes honeynets (HNA - *Honey Net Agent*) [18].

Outra preocupação foi a forma com que o IDS NIDIA interage com outros IDS ou Centros de Informações de Segurança para capturar informações externas de atualização em suas bases de dados. Um modelo de atualização foi proposto, para que o sistema tenha as suas estratégias atualizadas permitindo uma maior capacidade na tomada de decisão por parte dos agentes responsáveis. Para alcançar seu objetivo de atender as necessidades de informação dos IDSs parceiros, o sistema contará com a colaboração de cinco agentes: Interface (exibe uma interface entre o IDS e a Agência Central de Segurança), Gerador de Conexões (cria um arquivo texto com as conexões de ataque), *Surrogate* (cria uma representação interna no IDS dos arquivos de conexões), Construtor de SAARA (cria uma nova sociedade de agentes SAARA) e Monitor (cria constantemente requisições às bases de dados de ataques em busca de novas informações) [51].

Outro ponto importante dentro das inovações recebidas pelo NIDIA está relacionado à qualidade dos serviços prestados pelos agentes que compõe o NIDIA. É de extrema

importância, para tornar o IDS uma ferramenta confiável e de grande usabilidade, a garantia da qualidade dos serviços e a capacidade de recuperação quando do surgimento de um evento prejudicial ao sistema. Em [55], um mecanismo de tolerância a falhas é proposto. Este mecanismo é baseado em agentes que monitoram o sistema e que fornecem uma recuperação adequada para cada tipo de falha detectada. A sociedade de agentes proposta é composta por três agentes: Agente Sentinela (*System Sentinel Agent* - SSA), Agente de Avaliação de Falhas (*System Fault Evaluation Agent* - SFEA) e Agente de Replicação (*System Replication Agent* - SRA).

Grande parte destas atualizações, foi implementada utilizando-se a plataforma de desenvolvimento de agentes Zeus. Devido a esta ter sido a plataforma sobre a qual o NIDIA teve seu ciclo de desenvolvimento inicial. Para a atualização realizada por [55], teve-se a necessidade da migração de parte do sistema NIDIA para a plataforma JADE, devido a esta atualização necessitar de características intrínsecas a JADE, tais como uso dos modelos de comportamentos do JADE. Entre os modelos de comportamentos usados temos *CyclicBehaviour* e *OneShotBehaviour*.

A nossa proposta de atualização do NIDIA como IDS remoto também faz uso das funcionalidades da plataforma JADE, assim dando continuidade, todo o sistema NIDIA foi migrado. Atualmente o IDS NIDIA está funcionando sobre a plataforma de agentes JADE versão 3.2.

3.4 Comparação entre os IDS baseados em Agentes e o NIDIA

A Tabela 3.1 apresenta uma análise comparativa entre os sistemas de detecção de intrusão baseados em agentes apresentados no Capítulo 2 e o NIDIA (sem a funcionalidade de IDS remoto).

Protótipos de IDSs	**Método de Análise**		**Tipo de Análise**		**Tipo de Contra-Medida**		
	Abuso	**Anomalia**	**Rede**	***Host***	**Relatório**	**Manual**	**Automática**
Sparta	x	x	x	x	x	x	
AAFID	x		x	x			x
JAM		x		x	x	x	
Hummingbird	x	x		x	x	x	
IDA	x			x	x		x
NIDIA	x	x	x	x	x		x

Tabela 3.1 – Comparação entre os IDS baseados em Agentes e o IDS NIDIA

Assim, conforme exposto na Tabela 3.1, podemos perceber que o NIDIA utiliza as principais características de um IDS baseado em agentes para dar ao sistema uma poderosa capacidade de detecção, fazendo a análise por abuso ou anomalia, capturando dados da rede e de *hosts* e fornecendo resposta automática aos eventos acontecidos no ambiente a ser protegido através dos agentes.

3.5 Conclusões

Neste capítulo, apresentou-se o IDS NIDIA, suas características, sua arquitetura e seu funcionamento. Da sua arquitetura foi descrita a funcionalidade de cada camada e dos agentes pertencentes a cada uma delas. O funcionamento das principais atividades do NIDIA também foi detalhado.

Através do estudo do IDS NIDIA, podemos perceber a importância do uso de agentes no processo de detecção de intrusão. Apresentou-se as últimas alterações necessárias ao IDS NIDIA, para que este pudesse ser preparado para ser extendido com as funcionalidades de IDS Remoto.

4 Modelo do IDS-NIDIA Remoto

Originalmente, o NIDIA é um IDS para ser executado em um ambiente de rede local, ou seja, suas funcionalidades foram desenvolvidas de forma a prover os seus serviços de proteção e detecção de intrusos em uma rede local [36]. Sendo assim, o NIDIA é baseado em agentes dispostos em pontos estratégicos da rede local para prover uma maior capacidade de detecção.

Conforme exposto anteriormente, o foco dos ataques tem-se deslocado dos ambientes empresariais para os usuários finais. Um dos motivos que leva a esta mudança de alvo, é o crescente uso por parte das grandes empresas de um grupo de ferramentas (Firewall, Anti-Vírus, AntiSpaywares, Suites de Segurança, IDS) que provêem um ambiente de rede seguro e o pouco uso ou mesmo desconhecimento por parte dos usuários finais destas ferramentas de proteção. Apesar dos usuários finais terem iniciado uma reação a estes ataques, através do uso de ferramentas de segurança voltadas ao seu perfil (Firewall Pessoais, AntiVirus Pessoais, Ferramentas de Segurança embutidas no próprio sistema operacional do usuário final), este uso ainda é muito incipiente. Além disso, a preocupação geralmente é voltada para impedir que o ataque aconteça, mas quase nunca voltado a sua detecção após o atacante ter sucesso. Neste processo, com o objetivo de utilizar o poder da ferramenta IDS NIDIA para proteger não somente uma rede local, mas também os usuários finais, novos requerimentos e conceitos foram introduzidos em seu modelo e arquitetura [56].

A Figura 4.1 apresenta a nova proposta do IDS-NIDIA para fornecer um IDS Remoto. Nesta figura, apresentamos na extremidade direita o novo IDS-NIDIA com seus agentes distribuídos em uma rede local e com seus serviços expostos através da Internet. Na extremidade esquerda nós apresentamos os usuários externos que acessam os serviços do novo IDS-NIDIA através da Internet. Estes usuários externos poderão estar usando os mais variados dispositivos para se conectar ao serviço de IDS do novo IDS-NIDIA. No decorrer deste Capítulo apresentaremos os componentes e a interação entre estes para que o cenário apresentado na Figura 4.1 possa ser possível.

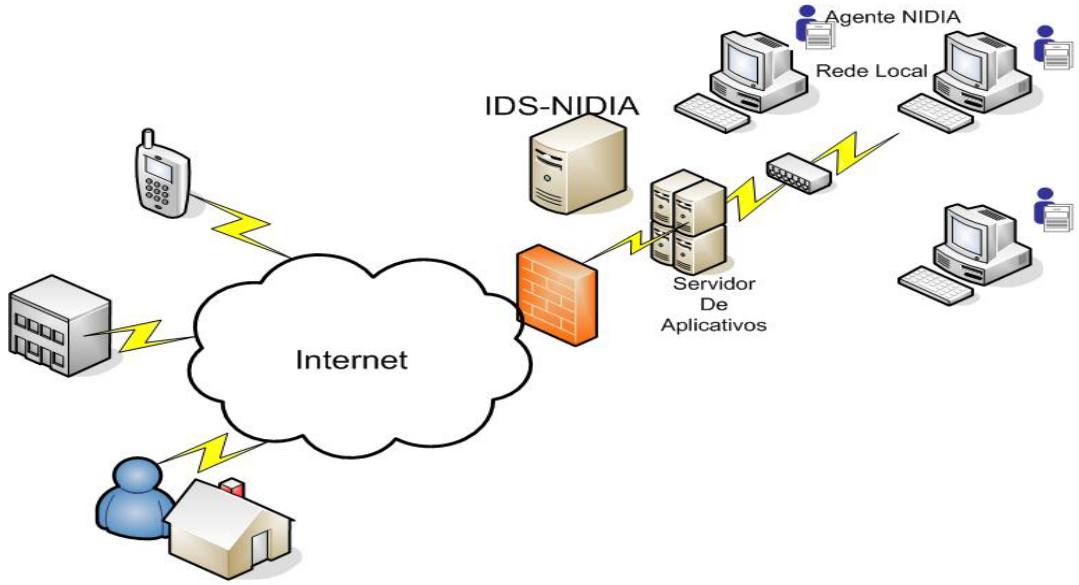

Figura 4.1: Modo de Funcionamento do novo IDS-NIDIA

Desta forma, os serviços de detecção do IDS-NIDIA foram adaptados e extendidos na forma de serviços dispostos na Internet. A proposta de nosso trabalho é prover uma ferramenta IDS distribuída com serviços remotos expostos na Internet.

4.1 Novos requerimentos e soluções para o NIDIA

Com a exposição dos serviços do IDS-NIDIA na Internet, um número maior de usuários será atingido em sua atividade de proteção. Devido a isto, o sistema foi modificado para estar habilitado a manipular um grande volume de informações (dados capturados das redes locais e dos usuários externos), proteger um número maior de usuários e detectar a maior variação de ataques possíveis. Uma característica do NIDIA é aprender com os ataques detectados. Uma vez que um ataque é detectado pelo NIDIA, a assinatura do ataque e as contramedidas usadas são armazenadas em bancos de dados para futura reutilização. Com a crescente utilização do sistema, a variação dos usuários será maior e conseqüentemente o aprendizado do sistema será maior. É certo que refinamentos serão necessários e o benefício deste refinamento será ter um sistema com maior capacidade de detecção e robustez suficiente para proteger os usuários de um número maior de ataques. Em outras palavras, o sistema NIDIA aprende com as imperfeições acontecidas em um ambiente A (uma rede local de uma empresa e/ou um usuário externo) para serem assimiladas e proteger um outro ambiente B (uma outra rede local e/ou um outro usuário externo). Para tornar este cenário possível, um conjunto de novos requerimentos se fez necessário:

- É necessário que os usuários finais tenham como enviar dados da sua máquina ou rede para o novo IDS-NIDIA e fazer uso deste, ou seja, é necessário o uso de um tipo de aplicação que uma vez colocada na máquina do usuário final, esta fica constantemente enviando informações do seu sistema ao IDS-NIDIA;
- É necessário um ponto de recepção disponível na Internet para receber as informações enviadas dos usuários finais ao IDS-NIDIA, ou seja, uma espécie de aplicação disponível na Internet, que tenha total disponibilidade para receber informações, formatá-las e enviá-las ao IDS-NIDIA para análise;
- É necessária, devido à heterogeneidade dos usuários finais, uma forma padronizada de manipular os dados a serem analisados e gerenciados pelo novo IDS-NIDIA, ou seja criar um modelo de dados padrão que possa ser compartilhado por todos os elementos envolvidos no uso do novo IDS-NIDIA;
- Devido o IDS-NIDIA original ser baseado em um sistema multiagentes, onde toda a sua funcionalidade está embutida no comportamento dos seus agentes, faz-se necessário criar uma "ponte" de comunicação dos itens externos ao IDS-NIDIA com os seus agentes, ou seja, a criação de uma camada de aplicação que permita aos agentes NIDIA exporem seus serviços, recebendo solicitações e retornando dados da sua execução para os itens externos ao IDS-NIDIA.

As novas soluções para os requerimentos do novo IDS-NIDIA são listados a seguir com uma breve descrição. Posteriormente, estes serão explanados de forma mais detalhada. Deste ponto em diante, nós utilizaremos o nome IDS-NIDIA Remoto para fazer referência ao novo IDS-NIDIA.

As novas soluções para o IDS-NIDIA Remoto:

- **IDS-Client** é um componente usado pelo usuário externo para permitir que este possa requisitar os serviços dispostos na Internet pelo IDS-NIDIA Remoto. Este componente é montado como um conjunto de pequenos módulos (*facility*) que são baixados e acoplados a uma interface padrão. Cada módulo possui uma atividade específica a desempenhar e a sua configuração é feita a partir das informações trocadas entre o usuário externo

e o ambiente de execução do IDS-NIDIA Remoto;

- **IDS-Proxy** é um componente desenvolvido através da tecnologia de web services, cujo principal objetivo é intermediar as interações entre o IDS-Client e a Camada de Gerenciamento Remoto. É ao IDS-Proxy que inicialmente chegam as requisições dos usuários externos. O IDS-Proxy recebe as requisições do IDS-Client, executa rotinas de tratamento necessárias e então encaminha as mensagens tratadas aos agentes da Camada de Gerenciamento Remoto;
- **NIDIA *Profiles*** são metamodelos criados para gerenciar as informações manipuladas pelo IDS-NIDIA Remoto. Uma das principais informações a serem gerenciadas pelo NIDIA *Profile* são os serviços remotos disponibilizados pelo IDS-NIDIA Remoto. O NIDIA *Profile* descreve os serviços que um usuário externo pode requisitar e como o sistema se adaptará para prover os serviços a este usuário. Os NIDIA *Profiles* podem também assistir na integração do NIDIA a outras ferramentas tais como Firewalls, antivírus ou Anti-Spywares, permitindo a um usuário montar um ambiente seguro e integrado;
- **Camada de Gerenciamento Remoto** é responsável por coordenar e gerenciar o trabalho em conjunto dos agentes NIDIA existentes no modelo do novo IDS-NIDIA com os novos agentes pertencentes a esta nova camada. Os novos agentes pertencentes a Camada de Gerenciamento Remoto em conjunto com o componente IDS-Proxy permitem que os dados externos possam ser encaminhados aos agentes internos do IDS-NIDIA Remoto. Da mesma forma, a combinação Camada de Gerenciamento Remoto e IDS-Proxy habilita o IDS-NIDIA Remoto a encaminhar seus dados internos aos usuários externos. É através desta troca de mensagens que o IDS-NIDIA Remoto tem seus serviços expostos na Internet. Esta camada auxilia os agentes do IDS-NIDIA Remoto, gerenciando a forma como estes serviços podem ser utilizados e como e quais serviços podem ser expostos pelos agentes internos do IDS-NIDIA Remoto. A Camada de Gerenciamento remoto auxilia também no uso dos NIDIA *Profiles*.

Um ponto importante que permitiu a implantação das soluções para os novos requerimentos do IDS-NIDIA Remoto foi a mudança da plataforma de desenvolvimento de

agentes ZEUS para a plataforma de desenvolvimento de agentes JADE. Pois estas soluções foram desenvolvidas fazendo-se uso da interação de três tecnologias: Sistemas Multiagentes, *Web Services* e MDA. Por exemplo, para permitir a interação dos agentes internos ao NIDIA com os agentes da Camada de Gerenciamento Remoto, através do IDS-Proxy, um *add-on* em Jade teve de ser implementado. Um *add-on* é um mecanismo de extensabilidade para o JADE.

O IDS-NIDIA Remoto tem seu ambiente montado através destes componentes que interagem entre si através do envio de mensagens com o objetivo de executar as tarefas do sistema. A Figura 4.2 apresenta a interação destes componentes previamente explicados. Na Figura 4.2, apresentamos o IDS-NIDIA Remoto com seus componentes e o IDS-Client instalado na máquina de um usuário externo. Em seu processo de execução um usuário externo através do IDS-Client solicita os serviços do IDS-NIDIA Remoto. Esta requisição do IDS-Client chega até o IDS-NIDIA Remoto através do IDS-Proxy. Uma das funcionalidades do IDS-Proxy é fazer com que essa requisição chegue até os agentes do IDS-NIDIA. Após a execução das tarefas dos agentes do IDS-NIDIA as informações voltam através do mesmo caminho de execução.

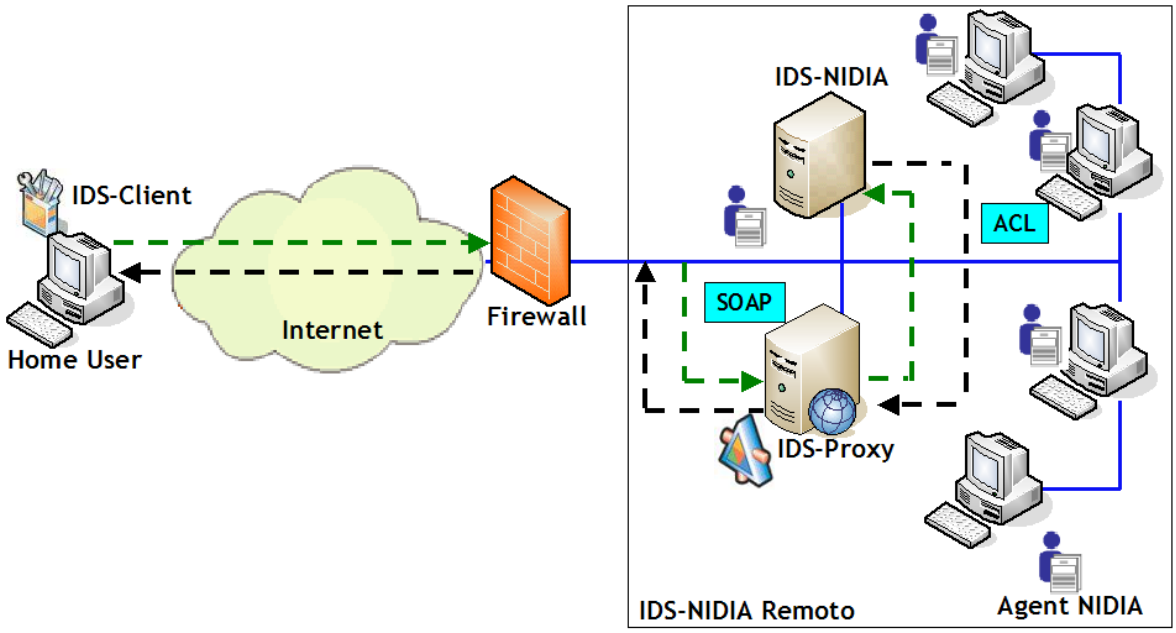

Figura 4.2: Interação entre os componentes do IDS-NIDIA Remoto

Na Figura 4.2, podemos perceber os componentes de software do sistema e suas dependências.

4.1.1 IDS-Client

O IDS-Client é uma aplicação cliente que é usada pelo usuário externo para procurar, escolher e acessar serviços remotos disponíveis pelo IDS-NIDIA Remoto. A Figura 4.3 apresenta os elementos fundamentais do IDS-Client.

O IDS-Client permite aos usuários externos acessarem e fazerem uso do IDS-NIDIA Remoto. O IDS-Client tem um relacionamento direto com o IDS-Proxy.

Uma vez que o IDS-Client está instalado na máquina do usuário, o sistema de diagnóstico obtém as informações da máquina, tais como espaço em disco, memória livre e tempo ocioso do processador. Essas informações são usadas pelo IDS-NIDIA para tomar a decisão de quais facilidades devem ser instaladas na máquina do usuário. Facilidades são elementos adicionais para o IDS-Client que fornecem funcionalidades tais como a interface básica ou gráfica para o usuário, captura de pacotes da rede, notificação ao usuário e execução das contramedidas. Após a instalação, o IDS-Client aciona o IDS-Proxy com o objetivo de criar um ambiente de execução remoto capaz de proporcionar o uso do IDS-NIDIA Remoto.

Uma Vez que a configuração está pronta, o IDS-Client pode iniciar a procura por ameaças na máquina do usuário fazendo uso das funcionalidades instaladas.

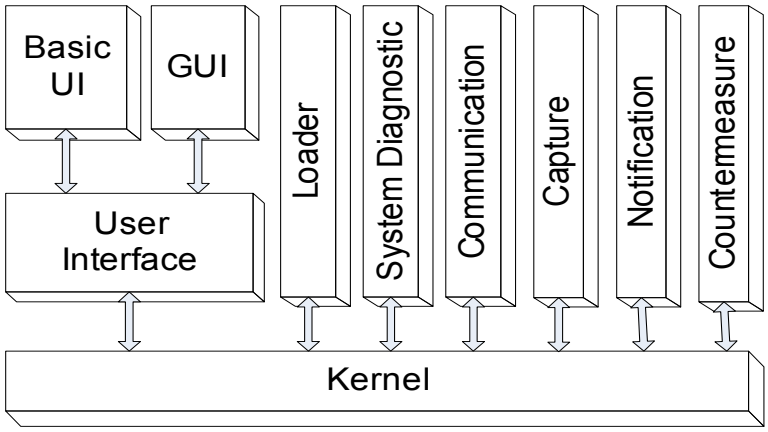

Figura 4.3: Estrutura do IDS-Client

Cada componente do IDS-Client é explicado como se segue:

- **User Interface**

É através da User Interface que o usuário interage com o sistema. É também através da User Interface que o sistema se comunica com o seu usuário, ou seja, este permite ao usuário obter informações visuais das ações que o IDS-Cliente está executando. A User Interface fornece as funcionalidades de apresentar ao cliente informações sobre o

diagnóstico do sistema, o status da comunicação com o IDS-Proxy (Ativa ou Inativa), o estado do processo de captura, as notificações que o usuário recebe e as contramedidas executadas. A User Interface, através da Basic UI, notifica o usuário na ocorrência de uma invasão;

- **Loader**

Este componente carrega as facilidades obtidas através do processo de diagnóstico realizado em conjunto com o IDS-NIDIA para identificação de quais os facilitadores serão "baixados" para o IDS-Client. Desta forma, o *Loader* inicializa os componentes facilitadores e mantém informações sobre o estado de cada um. O *Loader* também é responsável por manipular o NIDIA *Profile* para configuração interna dos módulos carregados. Esta configuração é feita através da leitura do NIDIA *Profile* e posteriormente aplicação das métricas definidas nele;

- **System Diagnostics**

Este módulo, como seu próprio nome indica, realiza o diagnóstico do Sistema, identificando o estado da máquina que terá o IDS-Client instalado. Assim, este módulo auxilia na tomada de decisão de quais facilitadores devem ser "baixados" no IDS-Client. Os dados obtidos são: o espaço livre em disco, espaço livre da memória e capacidade de processamento do processador;

- **Communication**

Este módulo é responsável por gerenciar a comunicação do IDS-Client com o IDS-Proxy. Este módulo abre a comunicação com o IDS-Proxy, suporta a troca de informações entre estes e fecha a conexão quando da necessidade do sistema não estar ativo (em um caso raro e extremo). O Módulo de comunicação envia o estado do processo de comunicação ao User Interface;

- **Capture**

Este módulo é responsável pelo processo de captura das informações trafegadas no sistema do usuário externo. Este é um processo multitarefa que enquanto uma linha de execução captura dados, uma outra linha de execução encaminha os dados obtidos para o

módulo de comunicação. Este módulo também trabalha em conjunto com módulo de Diagnóstico do Sistema, auxiliando na captura destas informações;

- **Notification**

Este módulo é responsável pela notificação dos eventos acontecidos durante o processo de execução do IDS-Client. É de responsabilidade do módulo de notificação apresentar através da User Interface informações sobre os ataques que o usuário possa estar sofrendo e as medidas que serão tomadas;

- **Countermeasures**

O módulo de contramedidas executa as ações repassadas ao IDS-Client pelo IDS-NIDIA Remoto através dos componentes IDS-Proxy e da Camada de Gerenciamento Remoto. A contramedida é o resultado das informações obtidas pelo IDS-Client e enviadas ao IDS-NIDIA Remoto para análise. Desta análise, é identificado um ataque e então IDS-NIDIA Remoto informa a estratégia de contramedida a ser executado pelo IDS-Client. Esta contramedida poderá ser executada exclusivamente pelo IDS-Client (exemplo, ativando um firewall local para fechar uma porta) assim como também, poderá ser executado em conjunto com o IDS-NIDIA Remoto (exemplo, fornecendo o *Honeypot*).

4.1.2 IDS-Proxy

O IDS-Proxy é uma camada *Web Service* dentro da arquitetura de camadas do novo IDS-NIDIA Remoto. O IDS-Proxy é um componente que em conjunto com a Camada de Gerenciamento Remoto permite ao usuário externo, através do IDS-Client, se beneficiar dos serviços do IDS-NIDIA Remoto. Assim, o IDS-Proxy dá ao sistema a capacidade de expor seus serviços na Internet.

Este componente é responsável por receber os dados de um cliente externo, enviados através de outro componente, o IDS-Client. O IDS-Proxy realiza a validação e formatação inicial das informações obtidas e então encaminha a informação tratada para o AgentProxy.

O AgentProxy é um *add-on* Jade que faz o link de comunicação do IDS-Proxy com a Camada de Gerenciamento Remoto. As mensagens enviadas ao IDS-Proxy via protocolo SOAP são convertidas em mensagens ACL entendidas pelos Agentes da Camada de Gerenciamento Remoto.

O IDS-Proxy fornece os serviços do IDS-NIDIA Remoto como serviços Web. Isto se deve pela necessidade dos usuários externos enviarem informações ao IDS-NIDIA Remoto para análise. Desta forma, IDS-Proxy funciona como um ponto de recepção de informação a todos os usuários que possuem um IDS-Client instalado em sua máquina e que estão conectados a Internet.

A Figura 4.4 apresenta a interação entre os componentes do IDS-Proxy. Na Figura 4.4 apresentamos as mensagens SOAP que chegam ao IDS-Proxy e são transladadas para mensagens ACL que então são manipuladas pelos agentes do IDS-NIDIA.

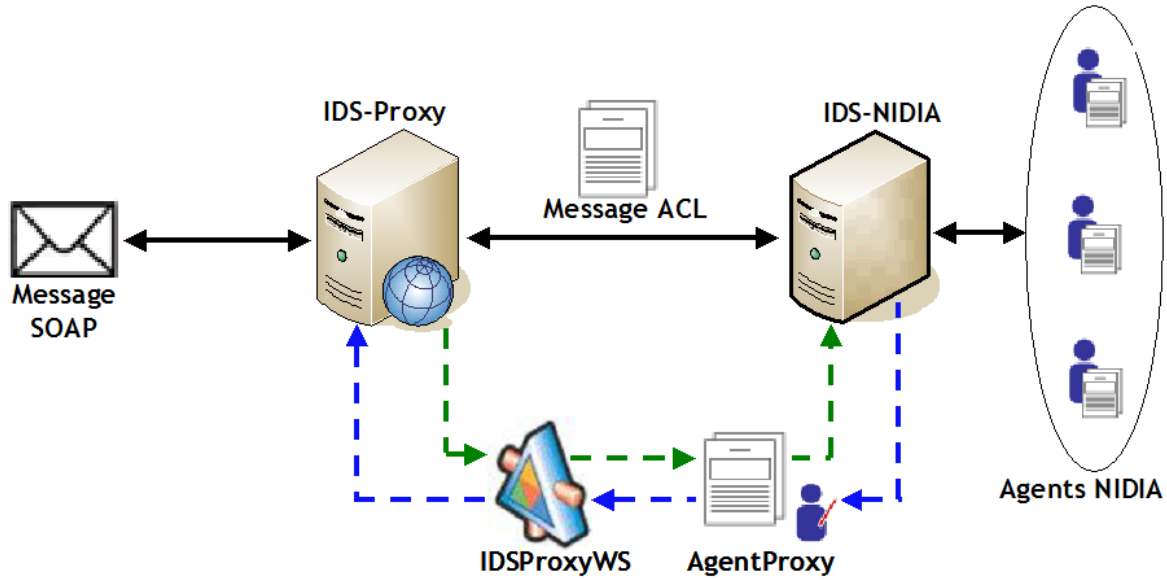

Figura 4.4: Interação entre os componentes do IDS-Proxy

O IDS-Proxy tem em seu desenvolvimento as seguintes características:

- É baseado na tecnologia dos *Web Services*;
- É responsável pela validação das informações obtidas dos IDS-Clients;
- Realiza uma formatação inicial dos dados obtidos através dos IDS-Clients;
- Mantém informação dos IDS-Clients conectados a ele;
- Utiliza as políticas de segurança e desenvolvimento definidas pela WS-I.

Outro componente importante dentro desta arquitetura é o AgentProxy.

O AgentProxy é responsável por fazer o link do IDS-Proxy, ponto de captura de informações dos usuários externos, aos agentes do IDS-NIDIA Remoto através da Camada de Gerenciamento Remoto. O AgenteProxy é baseado no código fonte do SocketProxyAgent. O SocketProxyAgent é um agente desenvolvido pela TILAB (JADE) [73], com o objetivo de permitir que informações externas à plataforma de um agente possam chegar até este. Este objetivo é alcançado disponibilizando *sockets* (*server* e *client*)

dentro do agente para receber informações da plataforma e para enviar informações da plataforma. O código fonte do SocketProxyAgent foi alterado e novas funcionalidades foram incluídas, tais como modificação do seu arquivo de configuração de nome <agentname>.inf de um simples arquivo texto a um arquivo em formato XML com informações sobre porta que o agente irá escutar suas mensagens, com quais agentes ele irá se comunicar, etc. Tais modificações foram realizadas para suprir as necessidades do IDS-NIDIA Remoto.

4.1.3 NIDIA *Profile*

O IDS-NIDIA Remoto faz uso de três tecnologias: Sistemas MultiAgentes, *Web Services* e MDA. O uso da tecnologia de Sistemas MultiAgentes, se dá devido a característica original do sistema NIDIA, ou seja por este ser um IDS baseado no uso de agentes inteligentes. O uso da tecnologia de *Web Services* fornece um suporte na interoperabilidade entre os diferentes componentes do sistema. O uso da tecnologia MDA é fundamental no gerenciamento de metadados, provendo a portabilidade, interoperabilidade e reusabilidade das informações.

A união destas tecnologias define um ambiente:

- **Inteligente**: fazendo uso dos agentes inteligentes, onde o conhecimento pode ser "programado" transformando-os em "peças de software" reativos;
- **Cooperativo:** fazendo uso dos web services como "atividade meio", cooperando na troca de mensagens entre os componentes clientes (IDS-Client) e os componentes servidores (IDS-NIDIA);
- **Integrado:** através do uso da informação compartilhada, ou seja, de conhecimento de todos os componentes do ambiente, porém integrada em um único repositório de metadados.

Este enfoque é compatível com as modernas tendências no desenvolvimento de sistemas accessíveis pela Internet.

A MDA provê o gerenciamento de metadados através dos modelos, metamodelos e transformações entre modelos usando conceitos e ferramentas baseadas em seus principais padrões UML, MOF e XMI. Os padrões MOF, JMI e XMI definem mecanismos comuns para a representação de metadados via APIs JAVA, serialização de metadados em XML e gerenciar o ciclo de vida dos metadados.

Assim, para prover o gerenciamento de metadados são previamente necessários alguns elementos, tais como:

- Um Metamodelo conforme ao MOF das informações a serem gerenciadas;
- Um arquivo XMI gerado a partir da transformação deste metamodelo;
- Um repositório de metadados onde o metamodelo será carregado através do arquivo XMI gerado;
- Rotinas para manipulação dos metadados através do repositório.

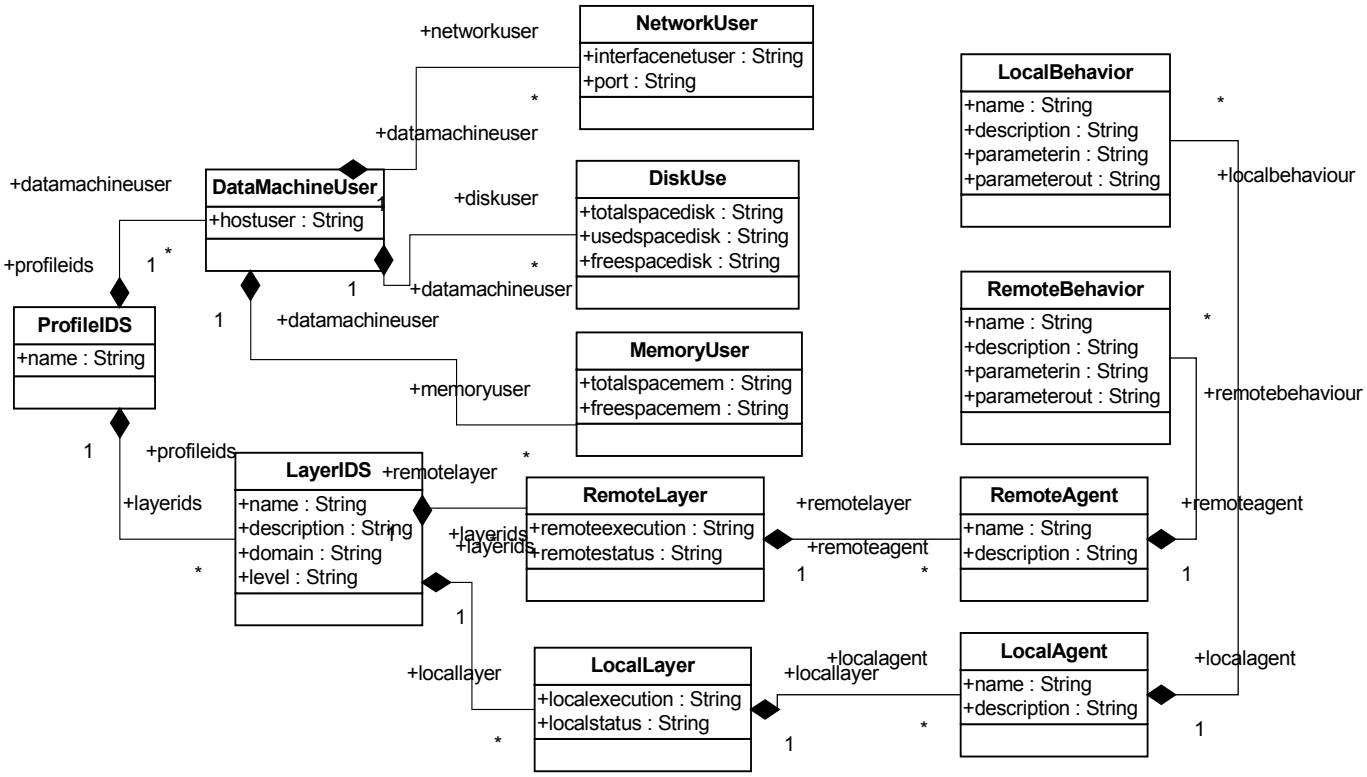

Figura 4.5: Metamodelo de Profile para o NIDIA

Os NIDIA *Profiles* contêm metamodelos para suportar o gerenciamento dos modelos (dados) do novo NIDIA. Estes metamodelos foram criados conforme a MOF. Um exemplo de informação a ser gerenciada através do NIDIA *Profile* pode ser a capacidade dos serviços remotos que o IDS-NIDIA Remoto provê para os usuários na Internet. A Figura 4.5 apresenta uma proposta do NIDIA *Profile*. No fragmento de diagrama apresentado na Figura 4.5 temos algumas informações manipuladas pelos componentes do IDS-NIDIA Remoto, tais como informações sobre o sistema do usuário final, sobre os agentes do IDS-NIDIA e seus serviços.

O Metamodelo da Figura 4.5 representa as informações armazenadas e manipuladas no repositório de metadados. A classe *DataMachineUser* e suas classes de composição,

representam as informações obtidas da máquina do usuário final. A classe *LayerIDS* e suas classes de composição, representam as informações sobre os serviços oferecidos pelas Camadas do IDS-NIDIA. Nestas classes de composição há informações sobre os agentes e seus comportamentos.

Os metamodelos para o NIDIA vêm dar a capacidade de gerenciar metadados tais como:

- Apresentar e manipular os dados dos serviços disponíveis para os usuários externos;
- Realizar o controle das informações a serem distribuídas entre os agentes que compõe o sistema;
- Gerenciar o controle da troca de Mensagens entre os componentes do Sistema;
- Manipular os dados capturados dos usuários externos.

Há um NIDIA *Profile* correspondente para cada item gerenciado e manipulado pelo IDS-NIDIA Remoto.

Para a criação dos NIDIA *Profiles* foram utilizadas as ferramentas Poseidon for UML Community Edition 3.2.0, MDR Explorer for NetBeans IDE 3.5.1 e UML2MOF. O uso destas ferramentas são explicados através dos passos da criação dos NIDIA *Profiles*.

Os passos para a criação dos NIDIA Profiles são os que se seguem:

1º Passo – Criação do Metamodelo através da Ferramenta Poseidon for UML Community Edition 3.2.0;

2º Passo – Conversão do arquivo XMI (Modelo UML) gerado pelo Poseidon for UML Community Edition 3.2.0 em formato XMI (metamodelo MOF), através da ferramenta UML2MOF;

3º Passo – Importar o arquivo XMI gerado para o Repositório MDR (MDR NetBeans) e gerar as interfaces JMI para manipulação do metamodelo;

4º Passo – Criação das rotinas de manipulação dos metadados. Rotinas de criação, inicialização e exclusão de instâncias de elementos do metamodelos.

Criação do Metamodelo através da Ferramenta Poseidon for UML Community Edition 3.2.0

Para a criação dos metamodelos na ferramenta Poseidon é necessário utilizar o *template TemplateForMetamodel.zargo*. Este *template* permiti gerar metamodelos conforme ao MOF. Para utilizá-lo é necessário abrí-lo no Poseidon e salvar o arquivo com

nome do metamodelo a ser criado. A Figura 4.6 apresenta um metamodelo criado através deste processo. No entanto, para que o metamodelo possa ser carregado no repositório de metadados, é necessário exportar o arquivo XMI gerado a partir deste metamodelo. Isto é feito no Poseidon escolhendo-se a opção de Exportar Projeto para XMI disponível no Menu Arquivo, tendo-se o cuidado de desmarcar a opção *Save with diagram data*.

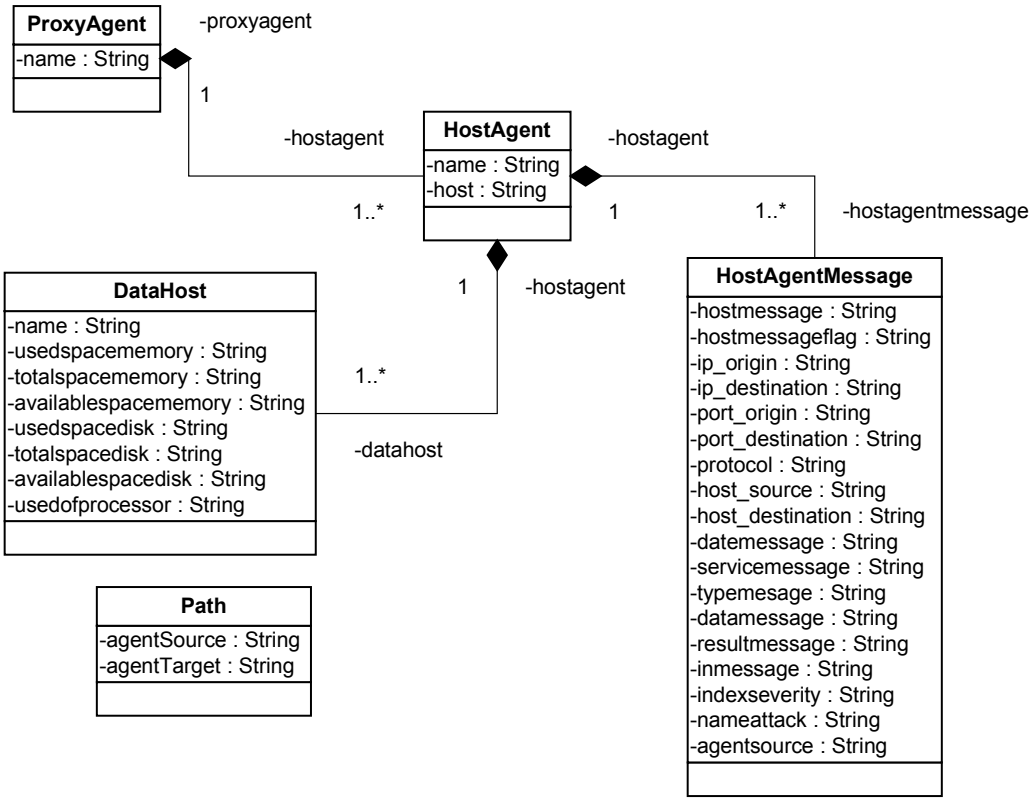

Figura 4.6: Metamodelo do Gerenciamento de Distribuição das Mensagens

Conversão do arquivo XMI gerado pelo Poseidon for UML Community Edition 3.2.0 em formato UML para o formato MOF, através da ferramenta UML2MOF

Apesar do arquivo XMI gerado no Poseidon 3.2.0 ser produzido a partir de um *template*, este por ser gerado em uma ferramenta com os padrões UML deve ser convertido em um formato MOF através da ferramenta UML2MOF.

A UML2MOF é uma ferramenta de linha de comando escrita em Java que tem dois parâmetros:

Parâmetro 1: o nome do arquivo XMI conforme à UML;

Parâmetro 2: o nome do arquivo XMI de saída. Este arquivo será conforme ao MOF.

A Figura 4.7 apresenta um trecho do arquivo XMI gerado para o metamodelo da Figura 4.6. Podemos perceber a relação entre as informações contidas no metamodelo da

Figura 4.6 e o trecho do arquivo XMI aqui apresentado. Na Figura 4.6, uma das classes pertencentes ao metamodelo é a classe *ProxyAgent* composta de um único atributo *name*. Podemos ver a relação destes elementos com as *tags* *Model:Class* e *Model:Attribute*, dispostas no arquivo XMI.

```
<?xml version="1.0" encoding="windows-1252" ?>
- <XMI xmi.version="1.2" xmlns:Model="org.omg.xmi.namespace.Model" timestamp="Sun Jul 16
    15:59:20 GMT-03:00 2006">
  - <XMI.header>
    - <XMI.documentation>
        <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
        <XMI.exporterVersion>1.0</XMI.exporterVersion>
      </XMI.documentation>
    </XMI.header>
  - <XMI.content>
      - <Model:Class xmi.id="a52" name="ProxyAgent" annotation="" isRoot="false"
          isLeaf="false" isAbstract="false" visibility="public_vis" isSingleton="false">
        - <Model:Namespace.contents>
          - <Model:Attribute xmi.id="a59" name="name" annotation=""
              scope="instance_level" visibility="public_vis" isChangeable="true"
              isDerived="false">
            - <Model:StructuralFeature.multiplicity>
                <XMI.field>1</XMI.field>
                <XMI.field>1</XMI.field>
                <XMI.field>false</XMI.field>
                <XMI.field>false</XMI.field>
              </Model:StructuralFeature.multiplicity>
            - <Model:TypedElement.type>
                <Model:PrimitiveType xmi.idref="a7" />
              </Model:TypedElement.type>
            </Model:Attribute>
      </Model:Package>
    </XMI.content>
  </XMI>
```

Figura 4.7: Trecho do arquivo XMI do Metamodelo da Figura 4.6

Importar o arquivo XMI gerado para dentro do Repositório MDR e gerar as interfaces JMI para manipulação do metamodelo

De posse do arquivo XMI gerado em conformidade com o MOF, este poderá ser então importado para o MDR.

Para realizar esta tarefa, as seguintes ações devem ser executadas:

- Instanciação do Modelo MOF;
- Importação do arquivo XMI conforme ao MOF;
- Geração das JMI Interfaces.

Deve-se portanto invocar o MDR *Browser* a partir do NetBeans IDE 3.5.1. Para isto, basta, a partir do menu *View* do NetBeans, escolher a opção MDR Browser, conforme apresentado na Figura 4.8.

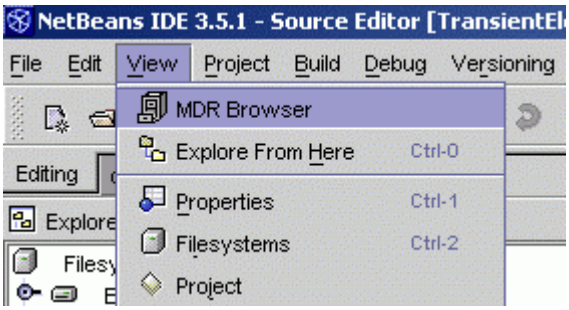

Figura 4.8: Invocação do MDR *Browser*

Com MDR *Browser* carregado, uma estrutura em arvore é montada cujo nó principal é o MDRepository. Abaixo do nó MDRepository, pode-se ver o nó MOF, o qual representa a instância de um modelo MOF (contendo metadados conforme ao próprio MOF). A partir deste nó MOF podemos criar instâncias MOF clicando com o botão direito do mouse sobre ele e escolhendo a opção "Instantiate" no menu pop-up que surge, conforme a Figura 4.9.

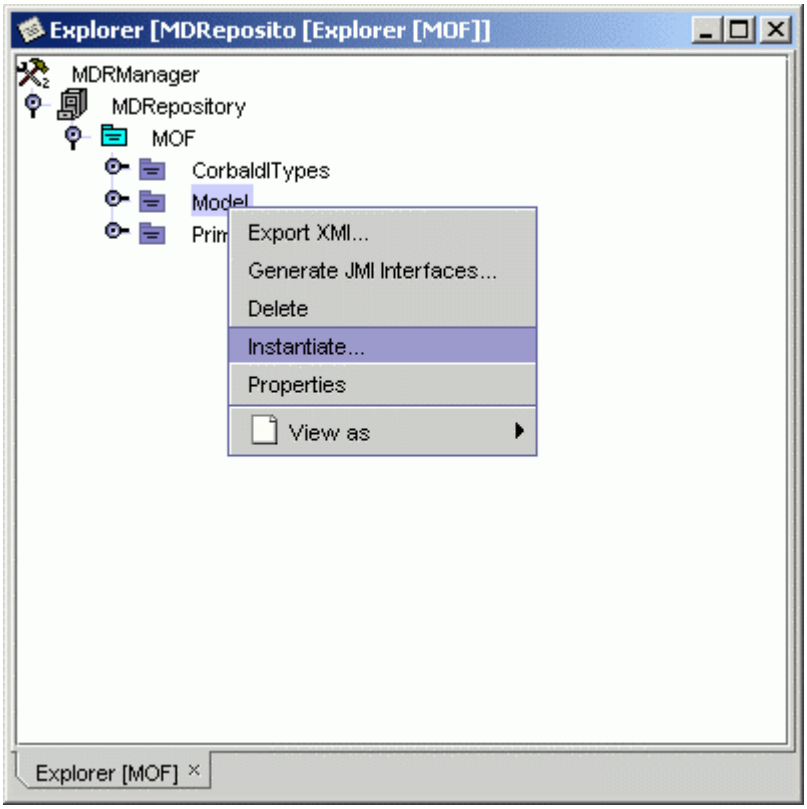

Figura 4.9: Instanciação do MOF

Uma caixa de diálogo aparece solicitando o nome da nova instância. Após informado, esta nova instância fica disponível na estrutura de árvore do MDR *Browser*, ficando no mesmo nível do nó MOF. O próximo passo é importar o arquivo XMI gerado no

passo anterior para dentro desta nova instância. Para isto basta clicar com o botão direito sobre a nova instância e escolher a opção "*Import* XMI", conforme a Figura 4.10.

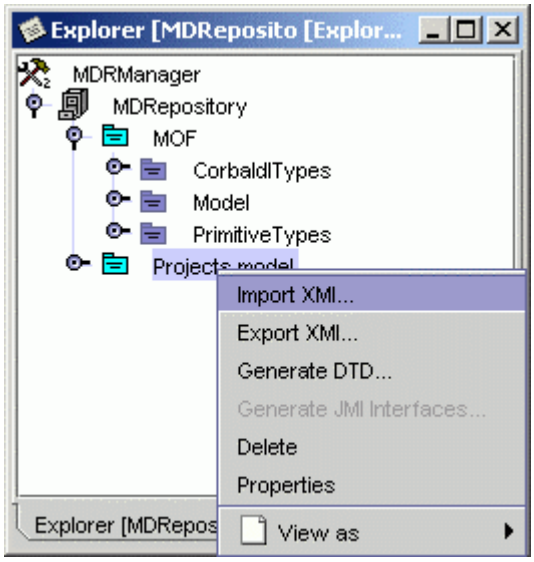

Figura 4.10: Importação do XMI

Desta forma, os elementos do modelo ficam dispostos dentro da instância criada. Para gerar as interfaces do metamodelo carregado no MDR *Browser*, pode-se clicar com o botão direito novamente na nova instância e no menu pop-up escolher a opção "*Generate* JMI *Interfaces*", conforme a Figura 4.11.

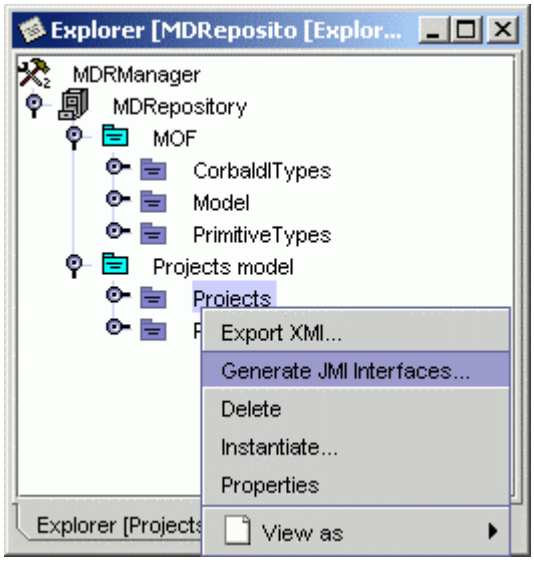

Figura 4.11: Geração das Interfaces

Neste passo, gera-se as interfaces para gerenciamento e acesso aos metadados do metamodelo, a partir de módulos externos. Para cada classe do modelo duas interfaces são

geradas – a primeira representando uma *factory* e a segunda representando instâncias individuais. As interfaces *factory* contêm métodos para criar novas instâncias de uma determinada classe e recuperar a lista de todas as instâncias existentes no repositório. As interfaces de instâncias contém métodos *getter* e *setter* para atributos e referências (isto é, Associações).

A Figura 4.12 apresenta o código Java gerado para uma Interface de Instância. Neste trecho de código, os métodos de acesso criados automaticamente pelo processo Geração de Interfaces no MDR *Explorer* são apresentados. De acordo com o metamodelo apresentado na Figura 4.6, a classe *ProxyAgent* possui um único atributo chamado *name*. Assim, métodos *get* e *set* são gerados para este atributo.

```
/**
 * ProxyAgent object instance interface.
 */
public interface ProxyAgent extends javax.jmi.reflect.RefObject {
    /**
     * Returns the value of attribute name.
     * @return Value of attribute name.
     */
    public java.lang.String getName();
    /**
     * Sets the value of name attribute. See {@link #getName} for
description
     * on the attribute.
     * @param newValue New value to be set.
     */
    public void setName(java.lang.String newValue);
    /**
     * Returns the value of reference hostagent.
     * @return Value of reference hostagent.
     */
    public java.util.Collection getHostagent();
}
```

Figura 4.12: Interface de Instância gerada pra a Classe ProxyAgent

Criação das rotinas de manipulação dos metadados. (Rotinas de criação, inicialização e exclusão de instâncias de elementos de metamodelos)

O principal benefício do MDR no desenvolvimento de softwares é oferecer aos desenvolvedores suporte à implementação de módulos que geram ou utilizam metadados. Isto torna mais fácil aos desenvolvedores criarem ferramentas ou módulos que interoperem com outras ferramentas.

Após a geração das interfaces, onde para cada elemento do metamodelo (pacotes, classes e associações) é gerada uma interface *factory* e de instância, faz-se necessário a criação de rotinas para a manipulação dos metadados, rotinas que permitam carregar

informações aos elementos do metamodelo no repositório MDR, ou seja, rotinas que permitam manipular o Repositório MDR.

Com esse objetivo, uma biblioteca escrita em Java 1.4 foi criada utilizando-se a ferramenta Eclipse Plataform 3.0.1. Esta biblioteca é responsável por:

- Abrir o repositório de metadados;
- Criar instâncias dos elementos do metamodelo dentro do repositório;
- Apagar instâncias dos elementos do metamodelo dentro do repositório;
- Atualizar as informações das instâncias dos elementos do metamodelo dentro do repositório;
- Fechar o repositório após a sua manipulação.

Esta biblioteca fornece também a funcionalidade de exportar as informações armazenadas no Repositório MDR para um arquivo XMI.

A Figura 4.13 apresenta um trecho do código desta biblioteca. Este trecho de código faz a abertura de um repositório chamado MDR disponível em "C:/modelNidia/mdr/" e a criação de um arquivo XMI baseado nos dados armazenados no repositório. Após a utilização do repositório MDR, a rotina o encerra através do método *shutdown*.

```
public class ExecuteDBAux {
	// name of a MOF extent that will serve as a target extent for the UML2MOF
transformation
	private static final String MY_INSTANCE = "pckDBAux_Inst";
	// name of a UML extent (instance of UML metamodel) that the UML models will be
loaded into
	private static final String My_MM = "pckDBAux";
	// repository
	private static MDRepository rep;
	// XMI reader
	private static XmiReader reader;
	// XMI writer
	private static XmiWriter writer;
	public static void main(String[] args) {
	try {
	System.setProperty("org.netbeans.mdr.persistence.Dir","C:/modelNidia/mdr/mdr");
		// look up an implementation of XmiReader interface
		reader = (XmiReader) Lookup.getDefault().lookup(XmiReader.class);
		// look up an implementation of XmiWriter interface
		XmiWriter writer = (XmiWriter) Lookup.getDefault().lookup(XmiWriter.class);
		// connect to the repository
		rep = MDRManager.getDefault().getDefaultRepository();
		. . .
		// close the repository
		rep.shutdown();
	}catch (Exception e) {
		ErrorManager.getDefault().notify(ErrorManager.ERROR, e);
	}
}
```

Figura 4.13: Trecho de código da biblioteca de manipulação do Repositório MDR

Os NIDIA *Profiles* podem ser acessados através de consultas ao repositório ou compartilhados através dos arquivos em formato XMI gerados a partir deste. Desta forma, as informações são transmitidas e compartilhadas entre os agentes que compõe o sistema e os demais componentes do IDS-NIDIA Remoto.

Através deste enfoque, os NIDIA *Profiles* provêem alguns benefícios tais como interoperabilidade, mobilidade e uniformidade de dados. Por exemplo, um usuário externo conhece as funcionalidades do IDS-NIDIA Remoto apenas acessando o NIDIA *Profile* que contém esta informação. Ou seja, um usuário que queira fazer uso dos serviços do IDS-NIDIA Remoto, não precisa saber nem entender como os serviços disponíveis estão implementados ou onde estão implementados, os usuários simplesmente fazem uso destes serviços através dos *Profiles*.

Os NIDIA *Profiles* também são utilizados no processo de configuração e no controle interno do IDS-NIDIA Remoto. Como exemplo, tem-se o NIDIA *Profile* de Controle de Distribuição de Mensagens entre os Agentes do IDS-NIDIA, cujo metamodelo é apresentado na Figura 4.6.

4.1.4 Camada de Gerenciamento Remoto

A Camada de Gerenciamento Remoto é responsável pela coordenação dos agentes que executam serviços remotos do IDS-NIDIA Remoto. Agentes que pertencem a esta camada realizam as seguintes atividades:

- Manipulam os *Profiles* através do repositório de metadados utilizados no processo de comunicação entre os componentes externos ao IDS-NIDIA Remoto e seus agentes. Esta é a camada responsável por manipular o NIDIA *Profile* preenchido com os dados fornecidos por um usuário ou auto-detectadas e, conseqüentemente, indicar quais agentes são necessários para executar as tarefas requisitadas.
- Selecionam os agentes que devem ser iniciados para a execução das tarefas solicitadas pelos usuários externos;
- Faz a "ponte" de comunicação entre os componentes externos ao IDS-NIDIA Remoto e seus agentes, transformando as requisições de serviços enviadas por usuários externos em mensagens de chamadas a comportamentos dos agentes NIDIA. Esta é a principal tarefa desempenhada por esta camada.

Esta camada é composta por seis agentes: agente de monitoramento remoto, agente de análise remota, agente remoto de reação, agente remoto de atualização, agente remoto de

administração e agente remoto de armazenamento. Cada agente da camada de gerenciamento remoto gerencia uma determinada camada do IDS-NIDIA Remoto. Desta forma a camada de gerenciamento remoto funciona como um elo de comunicação entre os agentes do IDS-NIDIA Remoto e seus usuários externos. O uso do IDS-Proxy é fundamental no uso da Camada de Gerenciamento Remoto como "ponte" de comunicação entre os componentes externos ao IDS-NIDIA Remoto e seus agentes. A Figura 4.14 apresenta a arquitetura do IDS-NIDIA Remoto.

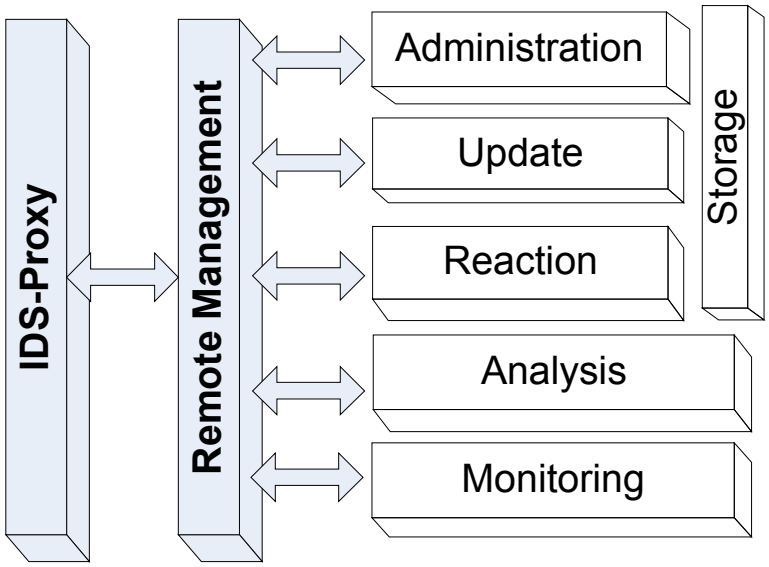

Figura 4.14: Nova arquitetura em Camadas do IDS-NIDIA Remoto

De acordo com a Figura 4.14, pode-se perceber a existência de um grande fluxo de comunicação entre os agentes da Camada de Gerenciamento Remoto e os agentes das demais camadas do IDS-NIDIA Remoto. Desta forma, para minimizar o *overhead* da comunicação, um conjunto de agentes foi desenvolvido para trabalhar como facilitadores e gerenciadores de comunicação entre as camadas.

Os agentes são:

- **Agente Capturador de Recursos** é responsável por obter informações sobre os recursos disponíveis na máquina, tais como espaço em disco, memória disponível e tempo ocioso do processador. Estes dados são posteriormente armazenados no repositório para que possa ser compartilhado com outros agentes, tal como o Agente de Balanceamento de Carga;
- **Agente Expresso** manipula o repositório de metadados, isto é, o repositório MDR montado durante a execução do IDS-NIDIA Remoto. É de

responsabilidade deste agente, prover aos demais agentes do sistema informações sobre os elementos pertencentes ao NIDIA *Profile* armazenados no repositório de metadados. O Agente Expresso tem conhecimento de todos os NIDIA *Profiles* armazenados no repositório de metadados;

- **Agente de Balanceamento de Carga** verifica se o sistema está em um estado de ponto de estagnação, e toma as medidas necessárias para manter a disponibilidade dos agentes da Camada de Gerenciamento Remoto. Este agente fica constantemente realizando consultas sobre as informações obtidas pelo agente de captura de recursos. Dados como espaço em disco, memória disponível e tempo ocioso do processador são utilizados como índices para sua tomada de decisão.

4.2 Novos conceitos para o NIDIA

Para prover os serviços de IDS na Internet, além das soluções vistas anteriormente, novos conceitos foram introduzidos aos agentes do IDS-NIDIA Remoto. Estes conceitos são introduzidos usando novos agentes, tais como os Agentes Wrapper e Agentes Host, e atualizando os agentes já pertencentes à arquitetura do antigo IDS-NIDIA. Os agentes já existentes são adaptados de forma a trabalhar em conjunto com os novos componentes do IDS-NIDIA Remoto. Os agentes foram implementados ou atualizados utilizando-se a Plataforma de agentes JADE. Os novos agentes são:

- **StartNIDIA** é responsável por iniciar a aplicação IDS-NIDIA Remoto, criando um agente LSIA (*Local Security Intelligent Agent*) no *main-container* e inicia o agente RMA (Remote Monitoring Agent);
- **LSIA** (*Local Security Intelligent Agent*) inicia os agentes do NIDIA através de um arquivo de configuração “agents.txt”. Ao iniciar o IDS-NIDIA Remoto, uma hierarquia de Agentes *Wrapper* contendo Agentes *Host* é criada. Este também é responsável por manter o repositório de metadados que é usado para manipular as informações dos agentes. O LSIA faz uso da biblioteca de manipulação do Repositório MDR;
- **NSA** (*Network Sensor Agent*) inicia o sensor de rede. Nesse agente, dois comportamentos são básicos: CapturePackage e SendPackage: CapturePackage inicia a captura de pacotes na rede, e o SendPackage pega os dados capturados e

envia-os para o agente SMA. Para este propósito, nós usamos a biblioteca Java Package for Packet Capture (JPCAP) [31];

- **SMA** (*System Monitoring Agent*) formata os dados que são coletados pelo NSA;
- ***Wrapper* Agente** intermedia a comunicação entre outros agentes do NIDIA. Cada agente do NIDIA (exceto o Agente *Wrapper*) tem um agente deste tipo correspondente. Este recebe as mensagens e distribui estas para outros agentes que estão menos carregados. Por exemplo, um agente SMA pode ter a ele associado diversos agentes SMAs que agem em variados *hosts*. Um Agente *Wrapper* associado ao agente SMA pode então receber as mensagens que chegam até este e encaminha para um SMA que esteja e melhor situação. Desta forma, o Agente *Wrapper* fornece um melhor uso do sistema, realizando o controle da distribuição das mensagens trocadas pelos agentes do sistema;
- **SEA** (*Security Evaluation Agent*) recebe mensagens do agente SMA, procede com a análise das mensagens, e envia uma mensagem para o MCA contendo a identificação da conexão e grau de severidade do ataque;
- **MCA** (*Main Controller Agent*) realiza o controle e o monitoramento das atividades dos outros agentes, devido a este agente, a qualquer momento, necessitar que os demais agentes do sistema se comportem de acordo com as estratégias definidas na STDB do NIDIA. Este recebe informação do agente SEA, de forma a iniciar a reação a um ataque ocorrido. Para este propósito, este requisita ao agente BAM para prover a estratégia de reação. De acordo com a estratégia de reação, o agente MCA requisita que outros agentes tomem as contramedidas face aos ataques;
- **BAM** (*Binary Association Memory*) realiza a identificação do tipo de ataque. Este recebe do agente MCA um vetor binário de dados e identifica o tipo de ataque. Este usa uma rede neural para identificar o tipo de ataque e retorna isto para o MCA;
- **SSA** (*System Sentinel Agent*) é responsável por detectar se um agente esta vivo ou não. Para este propósito, este envia mensagens de *ping* para outros agentes e aguarda o retorno da mensagem. Se um agente não responde e ocorre um *timeout*, então o SSA notifica o RMA que realiza a execução da estratégia de tolerância a falhas.

A seguir temos o modelo de comunicação para o IDS-NIDIA Remoto demonstrando como os componentes interagem para prover os serviços de IDS na Internet.

4.3 Modelo de Comunicação do novo IDS-NIDIA

A Figura 4.15 apresenta o modelo de comunicação para o IDS-NIDIA Remoto. Este modelo de comunicação pode ser detalhado de acordo com os seguintes passos:

- O IDS-Client requisita uma conexão com o IDS-Proxy (IDS-Client envia uma mensagem OpenConnection para o IDS-Proxy);
- O IDS-Proxy aceita ou recusa a conexão (IDS-Proxy retorna uma mensagem ConnectionResult para o IDS-Client);
- Se IDS-Proxy aceita a conexão, então o IDS-Client usa o componente *System Diagnostic* para obter a informação da máquina do usuário. Assim, o IDS-Client envia a informação da máquina (IDS-Client envia a mensagem SendDataMachine);
- O IDS-Proxy em conjunto com os agentes do IDS-NIDIA analisa as informações da máquina, seleciona e retorna outros componentes a serem usados no IDS-Client (IDS-Proxy retorna a mensagem FacilityComponents que contém os outros componentes);
- O IDS-Client requisita o NIDIA *Profiles* (IDS-Client envia a mensagem GetNIDIAProfiles). Então, o IDS-Proxy chama a rotina de Gerenciamento Remoto que, neste momento, requisita aos Agentes NIDIA suas capacidades. Uma vez que a Camada de Gerenciamento Remoto recebe as capacidades, este cria o NIDIA Profile e retorna este para o IDS-Proxy (IDS-Proxy envia a mensagem NIDIAProfile);
- O IDS-Proxy recebe o NIDIA-Profile e envia este para o IDS-Client (IDS-Proxy envia a mensagem NIDIA-Profile);
- O IDS-Client apresenta para o usuário os serviços IDS, e então o usuário seleciona os serviços IDS desejados. Após, o IDS-Client requisitar os serviços IDS selecionados, o IDS-Proxy requisita para a Camada de Gerenciamento Remoto criar um ambiente de IDS Remoto. Então, IDS-Proxy retorna os serviços IDS alocados para serem usados pelo IDS-Client;

- O IDS-Client captura dados tais como pacotes de rede, logs do host, e outras informações. Então, o cliente chama os serviços de IDS passando como paramento os dados capturados (IDS-Client envia a mensagem Call-IDS-Services). O ambiente de IDS Remoto executa os serviços requisitados, e retorna os resultados da análise (IDS-Proxy retorna a mensagem IDS-ServiceResult);

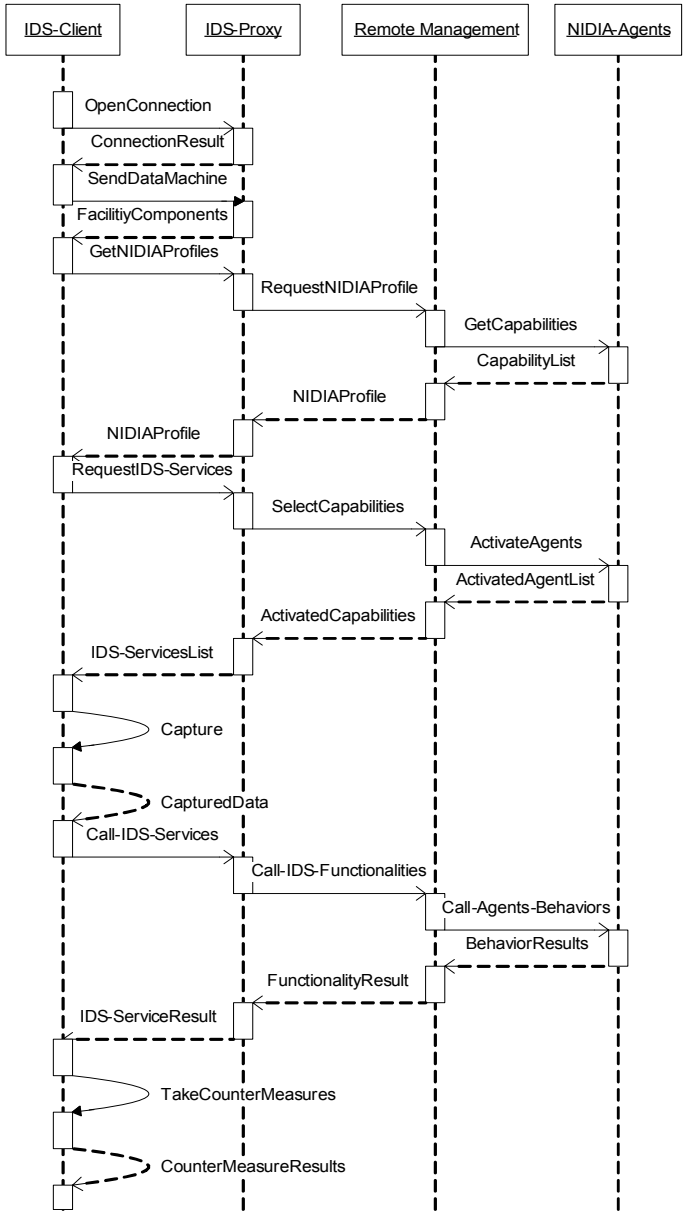

Figura 4.15: Modelo de Comunicação para o IDS-NIDIA Remoto

- O IDS-Client toma as contramedidas definidas pelo ambiente de IDS Remoto (IDS-Client envia a mensagem TomarContramedidas para ele mesmo). Para este propósito, o IDS-Client pode usar um componente de

contramedidas local ou requisitar suporte do ambiente de IDS Remoto. Por exemplo, no caso de um ataque DoS (Denial of Service), o IDS-Client pode apenas usar o componente de contramedidas local para fechar uma conexão com um *host* malicioso. No caso de um ataque de um *Worm*, o componente de contramedida pode requisitar suporte para o ambiente de IDS Remoto para prover uma estratégia de *honeypot* face a este tipo de *worm*. Uma vez que o IDS-Client finalize a execução das contramedidas, o IDS-Client apresenta os resultados para o usuário.

4.4 Conclusão

Neste capítulo, os requisitos necessários para o desenvolvimento de um IDS-Remoto foram apresentados. Desta forma, as soluções necessárias para que o Modelo proposto possa ser implementado foram discutidas. Apresentamos também as modificações necessárias ao IDS-NIDIA para que este pudesse ser utilizado como a ferramenta IDS-Remoto. Finalizamos o capítulo, apresentando o fluxo de comunicação entre os componentes pertencentes ao modelo. O próximo passo é a implementação de protótipos que possam demonstrar de forma prática, a utilização e aplicação do IDS-NIDIA Remoto. O próximo capítulo explana a respeito da implementação dos protótipos, aqui teoricamente apresentados.

5 Prototipagem do IDS-NIDIA Remoto

Com o propósito de realizar testes e mensurar resultados sobre o modelo proposto no Capítulo 4, foram implementados protótipos de cada componente pertencente ao modelo. As implementações seguem todas as especificações e incorporam todos os módulos definidos pela arquitetura proposta. Neste capítulo, detalhes das implementações destes protótipos serão apresentados.

5.1 Introdução

Os protótipos desenvolvidos demonstram o uso do modelo do IDS-NIDIA Remoto. Este modelo é baseado no uso de cinco elementos, conforme apresentado no Capítulo 4: o IDS-Client, o IDS-Proxy, a Camada de Gerenciamento Remoto, os Profiles de Utilização e um Serviço de IDS. Os módulos foram desenvolvidos de forma independente, porém com a capacidade de integração necessária inerentes ao seu uso. Esta estrutura possibilita o monitoramento de redes locais e remotas (usuários finais conectados a Internet).

Os protótipos foram desenvolvidos na linguagem Java e utilizam a plataforma de agentes JADE, *Web Services* e MDA. Estes protótipos adequam o ambiente de programação de acordo com a necessidade da implementação.

O processo aplicado para o desenvolvimento, bem como as características de implementação e ferramentas utilizadas são descritas nas seções subseqüentes.

5.2 Implementação do Protótipo

Nós iniciaremos a explicação do desenvolvimento dos protótipos, tecendo alguns comentários sobre as ferramentas utilizadas. Após, apresentamos o desenvolvimento de cada componente do Modelo através de Diagramas UML, *screenshots* de Telas e trechos de código-fonte.

Três ferramentas se destacam dentro do desenvolvimento dos protótipos. São elas:

- Plataforma de Desenvolvimento de Agentes JADE [65];
- *Systinet Developer for Eclipse* 6.0 [62];
- MDR *Browser for NetBeans IDE 3*.5.1 [37];

A Plataforma de Agentes JADE e o MDR *Browser for NetBeans IDE 3*.5.1 já foram inicialmente apresentados no Capitulo 2, daremos aqui uma visão voltada à programação com estas ferramentas. A ferramenta *Systinet Developer for Eclipse* 6.0 foi utilizada no

desenvolvimento do IDS-Proxy e da facilidade *Communication* do IDS-Client (camada de *Web services*).

5.2.1 Ferramentas Utilizadas

Plataforma de Agentes JADE

A arquitetura da plataforma JADE é baseada na coexistência de várias JVM (*Java Virtual Machine*) podendo ser distribuída por diversas máquinas independente de sistema operacional que cada uma utiliza. Na Figura 5.1, nós temos uma visão distribuída da plataforma de agentes JADE dividida em 3 *hosts.*

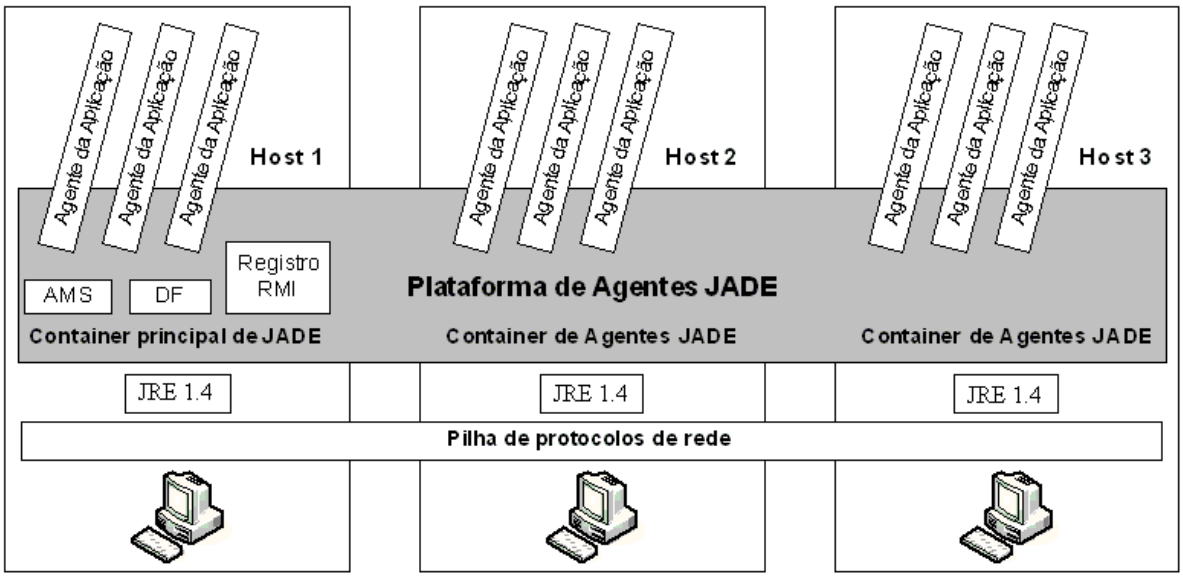

Figura 5.1: Plataforma de Agentes JADE

Em cada *host* temos uma JVM, JRE 1.4 (*Java Run-time Enviroment,* versão 1.4*)* para enfatizar o conceito de independência de plataforma. Em cada JVM temos basicamente um *container* de agentes que fornece um ambiente completo para execução destes agentes, além de permitir que vários agentes (cada agente é representado na Figura 5.1 pelo quadro Agente de Aplicação*)* possam rodar concorrentemente no mesmo processador (*host)*. Assim, durante a execução, deve existir uma JVM por processador, sendo possível vários agentes por JVM. A comunicação entre JVMs é feita através de RMI (*Remote Method Invocation)* de Java.

O JADE *Main-Container,* localizado no *Host 1* da Figura 5.1, é o container onde se encontra o AMS (*Agent Management System*), o DF (*Directory Facilitator*) e o registro

RMI (*Remote Method Invocation Registry)*. Esse registro RMI nada mais é que um servidor de nomes que Java usa para registrar e recuperar referências a objetos através do nome. Ou seja, é o meio que JADE usa em Java para manter as referências aos outros *containers* de agentes que se conectam a plataforma. Os outros *containers* de agentes conectam ao JADE *Main-Container* fazendo com que o desenvolvedor fique abstraído da separação física dos *hosts*, caso exista, ou das diferenças de plataformas dos quais cada *host* possa ter. Essa abstração é ilustrada na Figura 5.1, na parte central, na qual a plataforma JADE integra todos os três *hosts* atuando como um elo de ligação e provendo um completo ambiente de execução para qualquer conjunto de agentes JADE.

Um agente JADE funciona como uma *"thread*" que emprega múltiplas tarefas ou comportamentos e conversações simultâneas. Esse agente JADE é implementado como uma classe Java chamada *Agent*.

Do ponto de vista do programador, um agente JADE é simplesmente uma instância da classe *Agent*, no qual os desenvolvedores deverão escrever seus próprios agentes como subclasses de *Agent*, adicionando comportamentos específicos de acordo com a necessidade e objetivo da aplicação, através de um conjunto básico de métodos. A Figura 5.2 apresenta um fragmento das duas principais classes que dão suporte na criação de um Agente JADE.

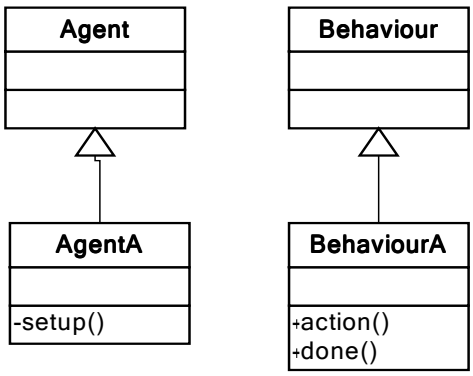

Figura 5.2: Fragmento das Classes de um Agente JADE

Cada serviço/funcionalidade de um agente deve ser implementado como um ou mais comportamentos. A abstração de comportamento do modelo do agente de JADE permite a integração de softwares externos para enriquecer a arquitetura do agente.

Todo e qualquer comportamento de um agente JADE é na verdade uma subclasse da classe *Behaviour*. A Classe *Behaviour* é uma classe abstrata do JADE que tem como finalidade provê a estrutura de comportamentos para os agentes JADE.

O desenvolvedor que pretende implementar um comportamento deve escrever uma ou mais subclasses *Behaviour,* instanciá-las e adicioná-las ao agente.

No nosso protótipo, nós utilizamos o JADE versão 3.2[14].

Systinet Developer for Eclipse 6.0

Systinet Developer for Eclipse é um *plug-in*, um módulo auto-contido que se acopla ao IDE do *Eclipse*. *Systinet Developer for Eclipse* permite ao desenvolvedor criar, testar, disponibilizar e gerenciar aplicações baseadas em *Web service* seguindo os padrões e regras do IDE *Eclipse*.

O desenvolvimento de *Web services* usando *Systinet Developer for Eclipse* é semelhante ao desenvolvimento de uma classe Java qualquer, conforme podemos ver na Figura 5.3.

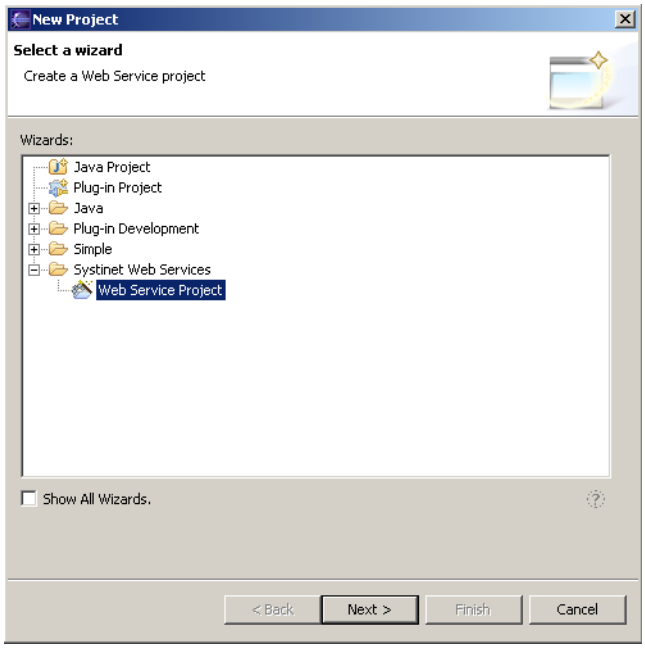

Figura 5.3: Tela de Novo Projeto do Eclipse.

Junto com o *plug-in* da *Systinet* é fornecido um *Systinet Server for Java*. Um servidor não seguro, por *default*, utilizado para o *deploy* e testes dos *Web services* criados.

Quando o desenvolvimento de um *Web service* é finalizado, o desenvolvedor pode registrar seus serviços no UDDI *registry*.

[14] O JADE versão 3.2 pode ser obtido em http://jade.tilab.com/.

Systinet Developer for Eclipse permite criar clientes que remotamente chamam os serviços disponibilizados em um servidor. A ferramenta permite fazer o *debug* remoto e também capturar mensagens SOAP passadas entre as aplicações cliente e servidor.

Systinet Developer for Eclipse possui como características principais:

- Geração automática do WSDL a partir de Classes Java e geração de Classes Java a partir de arquivos WSDL;
- SOAP *Spy*: uma ferramenta que captura mensagens SOAP entre o cliente e o servidor;
- *Systinet Server for* Java um servidor que dá suporte ao gerenciamento e *deploy* dos *Web services* criados;
- UDDI *integration*: suporta as especificações do UDDI versão 2 e 3.

Systinet Developer for Eclipse versão 6.0 suporta WS-I através do *Web services Interoperability Compliance Check*. Para fazer uso de WS-I *Compliance Checking*, deve-se fazer *download* do *Systinet* WS-I *Validator Plug-in*.

Systinet Developer for Eclipse através do *Systinet Server for* Java provê suporte a vários mecanismos de segurança, tal como SSL (*Secure Socket Layer*) com autenticação HTTPBasic. Usando a característica de segurança avançada do *Systinet Server for* Java, pode-se:

- Criar e rodar um *Systinet Server for* Java em modo seguro;
- Criar e fazer o *deploy* de um *Web service* seguro;
- Criar um usuário e dar permissões a ele;
- Criar um cliente seguro.

Conforme a Figura 5.4, podemos observar um servidor *Systinet Server for* Java não seguro e um servidor *Systinet Server for* Java seguro disponíveis para utilização pelo desenvolvedor.

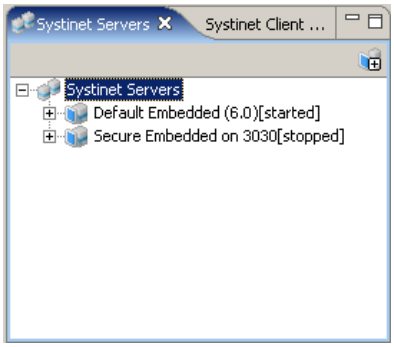

Figura 5.4: *Systinet Servers*

O *Systinet Developer for Eclipse* versão 6.0 pode ser obtido no site da Systinet[15].

MDR *Browser for NetBeans IDE 3*.5.1

O MDR *Browser for NetBeans IDE 3*.5.1 foi coberto de forma introdutória, na explicação relacionada ao seu uso no desenvolvimento dos NIDIA Profiles, no Capítulo 4. Reafirmamos aqui a importância de sua utilização e suas principais características que definem o "porquê" da sua utilização.

O MDR implementa o padrão MOF (*Meta Object Facility*) da OMG (*Object Management Group*). O MDR é baseado em padrões industriais definida pela OMG e JCP (Java *Community Process*). O MDR é integrado à Plataforma de Ferramentas do *NetBeans*, e contém a implementação de um repositório MOF incluindo um mecanismo de armazenamento persistente para metadados. A interface do repositório MOF é baseada no JMI (*Java Metadata Interface*). O MDR também define características que ajudam a incorporá-lo no IDE *NetBeans* (exemplo, mecanismo de notificação de eventos). O MDR é mais que uma infra-estrutura de um projeto, conforme o apresentado no Capitulo 4. Para apresentar a funcionalidade de um MDR, é usualmente necessário escrever um módulo que use o MDR.

Toda a documentação técnica, assim como o *download* do MDR *Browser for NetBeans IDE 3*.5.1 estão disponíveis no site da *NetBeans*[16].

5.2.2 Implementação do IDS-Client

O IDS-Client é um componente que possui três funções principais:

- Capturar informações do cliente;
- Enviar as informações capturadas ao IDS-NIDIA Remoto;
- Receber e executar as contramedidas indicadas pelo IDS-NIDIA Remoto após análise.

Um protótipo do IDS-Client foi desenvolvido para demonstrar a execução das suas três principais funcionalidades.

Na Figura 5.5, nós apresentamos o diagrama de classes em UML do protótipo do IDS-Client.

[15] http://www.systinet.com/.
[16] http://mdr.netbeans.org/.

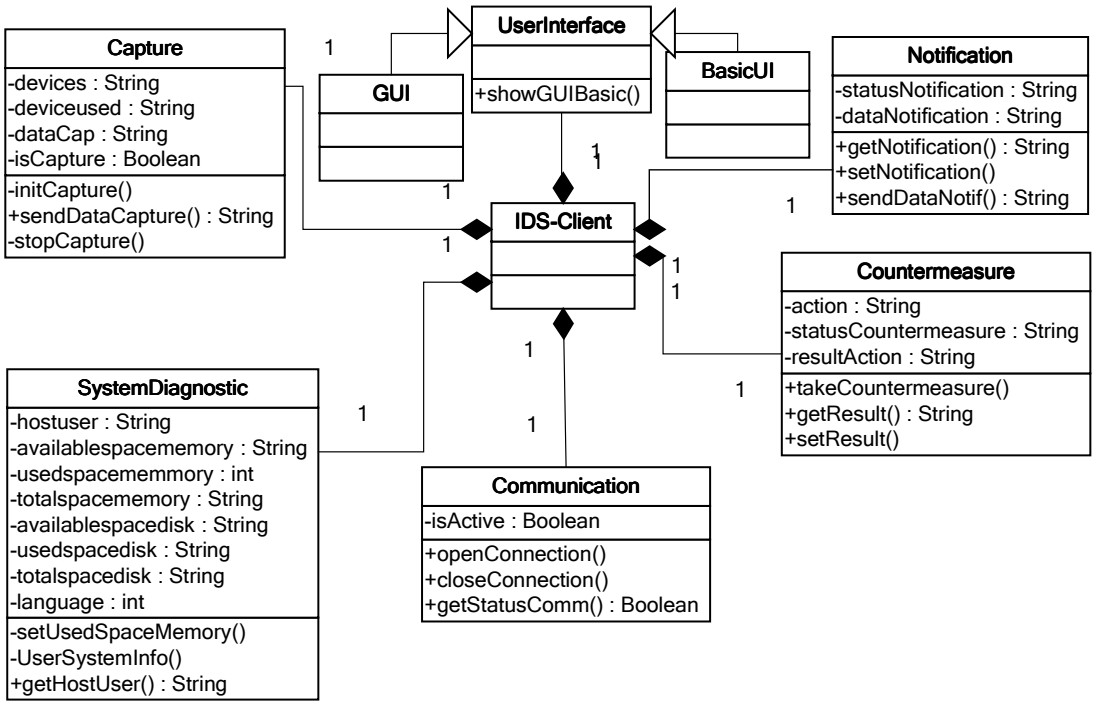

Figura 5.5: Diagrama de Classes do Protótipo IDS-Client (fragmento)

Conforme podemos observar, o Protótipo é composto de 9 Classes, sendo que a Classe IDS-Client é a classe principal dentro do modelo. Esta classe representa o Protótipo do IDS-Client e cada classe que a compõe representa uma *facility*. Estas *facilities* são agrupadas em arquivos *jar* que são configuradas e integradas ao ambiente do usuário final.

A partir deste diagrama de classes, um protótipo foi desenvolvido utilizando-se o IDE Eclipse 3.1 e a linguagem de programação Java versão 1.4.2. A Figura 5.6 apresenta um *screenshot* do Protótipo.

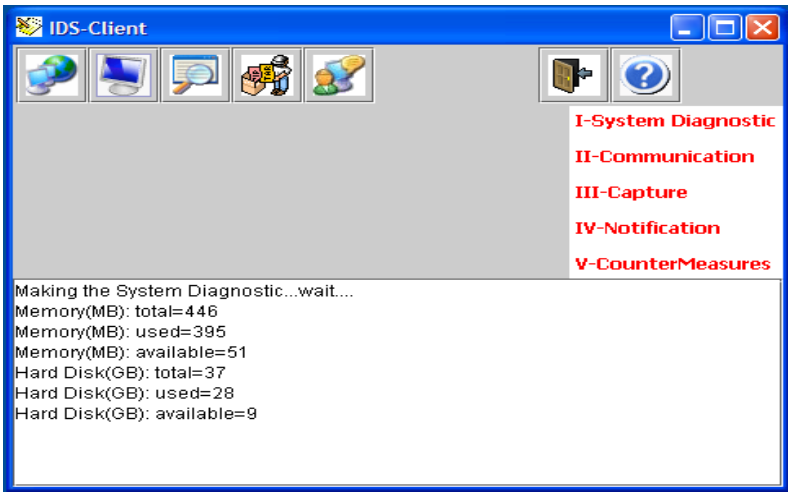

Figura 5.6: Tela Principal do IDS-Client

A estrutura de projeto do Protótipo é que se segue:

- Pacote br.nidia.idsclient.facility:
  - Capture.java;
  - Common.java;
  - GUIBasic.java;
  - SystemDiagnostic.java;
  - Communication.java;
  - Notification.java;
  - Countermeasure.java.
- Pacote br.nidia.idsclient.start:
  - StartIDSClient.java;
- Resources:
  - Imagens utilizadas na tela principal do IDS-Client.
- Bibliotecas Proprietarias:
  - JConfig.zip;
  - Jpcap.jar.

Segue uma explanação do protótipo através do seu pacote principal e de algumas classes que o compõe. É explanado também sobre a relação das classes com as bibliotecas proprietárias.

Package br.nidia.idsclient.facility

Este pacote agrega o conjunto de classes responsáveis pelas principais funcionalidades desempenhadas pelo IDS-Client.

Class Capture.java

Esta classe representa a *facility Capture*. Esta classe executa uma *Thread* que realiza as seguintes atividades:

- Identifica as interfaces de rede existentes;
- Define a interface de rede a ser usada;
- Inicia a captura dos dados trafegados;
- Envia os dados capturados;
- Encerra a captura dos dados.

A biblioteca *Jpcap* é utilizada para auxiliar no processo de captura. Esta biblioteca auxilia no processo de descoberta das interfaces de rede existentes e no processo de captura

dos dados trafegados nelas. O trecho de código na Lista de Código 5.1 apresenta o método construtor da classe *Capture* e o método *startCaptureThread* que inicia a *Thread* responsável pela captura dos dados de uma dada interface de rede.

```
public class Capture {
        public Capture(){
                this.filter = "ip and tcp";
                this.deviceused = 2;
                this.devices = Jpcap.getDeviceList();
                this.Datacap = "";
                this.isCapture = false;
        }
        /**
         * This Method init the capture. Start the captureThread.
         * The captureThread it makes the capture and it processes
         * the packages of parallel form.
         */
        private void startCaptureThread(){
                if (captureThread != null){
                        return;
                }
                captureThread = new Thread (new Runnable(){
                        public void run(){
                                while (captureThread != null){
                                        if (jpcap.processPacket(1,handler) == 0 && !isCapture){
                                                stopCaptureThread();
                                        }
                                        else{
                                                Thread.yield();
                                        }
                                }
                                jpcap.close();
                        }
                });
                captureThread.setPriority(Thread.MIN_PRIORITY);
                captureThread.start();
        }
}
```

Lista de Código 5.1: Trecho da Classe *Capture.java*

Class SystemDiagnostic.java

Esta classe representa a *facility SysmtemDiagnostic*. Esta classe executa a captura das informações da máquina do usuário final o qual o IDS-Client está instalado. As informações capturadas estão relacionadas à performance do seu sistema, dados como:

- Espaço em memória disponível;
- Espaço em memória usado;
- Espaço em disco rígido disponível;
- Espaço em disco rígido usado;
- Tempo ocioso do processador.

A Classe *SystemDiagnostic.java* possui dois métodos principais:

- Método *UseSystemInfo*: este método executa a captura das informações sobre o uso da memória e do processador no sistema do usuário final, através da execução do comando *systeminfo* do sistema operacional Windows XP. Por estar associado a um comando proprietário do sistema operacional, a classe antes de sua execução, identifica qual sistema operacional está sendo executado na máquina do usuário final e qual a linguagem do sistema.

Foram feitos alguns testes usando os comandos free[17] e df-h[18], em um sistema Linux, para obtenção das informações sobre o sistema do usuário final;

- Método *InformationHD*: este método captura as informações dos discos rígidos usados pelo sistema do usuário final. Este método faz uso de uma biblioteca proprietária chamada *JConfig*, que auxilia na montagem dos discos existentes e na captura das informações sobre os mesmos.

O trecho de código na Lista de Código 5.2 apresenta uma parte do método *UseSystemInfo*.

```
/**
 * This Method makes the verification of information of the system an informed
 * language. The System Diagnostic execute the command systeminfo for capture
 * informations in the system. This search information through an informed key.
 */
private String UseSystemInfo(String searchkey){
        int sep = 0;
        Process p;
String sItemobject = "";
String sItemdata = "";
try {
    String linha;
    p = Runtime.getRuntime().exec("systeminfo");
    BufferedReader input = new BufferedReader (new InputStreamReader(p.getInputStream()));
    while ((linha = input.readLine()) != null) {
        sep = linha.indexOf(':');
                if (sep != -1){
                        sItemobject = linha.substring(0,sep);
                        if (sItemobject.equals(searchkey)){
                                sItemdata = linha.substring(sep+1,linha.length()).trim();
                                break;
                        }
                }
         }
    input.close();
    }
   catch (Exception err) {
    err.printStackTrace();
    }
p = null;
        return sItemdata;
}
```

Lista de Código 5.2: Trecho do método *UseSystemInfo* (fragmento)

Class Communication.java

Este Classe representa a *facility Communictaion*. Esta classe é responsável por realizar as seguintes atividades:

- Abrir a comunicação com o IDS-Proxy;
- Auxiliar o processo de comunicação entre o IDS-Client e o IDS-Proxy, informando os *status* das atividades que estão sendo realizadas (Comunicação Ativa, Dados Enviados, etc);
- Enviar e receber dados do IDS-Proxy;
- Encerrar a comunicação com o IDS-Proxy.

[17] O comando free exibe a memória livre, a usada e os *buffers* da memória RAM.
[18] O comando df-h exibe o espaço em disco das partições montadas.

A classe *Communication.java* é um cliente do *Web service* desenvolvido no IDS-Proxy. A classe *Communication.java* é responsável por conectar o IDS-Client ao IDS-Proxy, fazendo uso do serviço de recebimento de dados disponibilizado pelo IDS-Proxy. Para efetuar a conexão ao IDS-Proxy e habilitar o canal de comunicação, a classe *Communication.java* se utiliza do método *initCommunication*. Este método descobre o serviço do IDS-Proxy obtendo informações sobre a *URI* da WSDL e do Serviço. O método *initCommunication* obtém uma instância do serviço disponível pelo IDS-Proxy e esta instância será utilizada pelos métodos de envio e recebimento de dados pertencentes também a classe *Communication.java*. O trecho de código na Lista de Código 5.3 apresenta o método *initCommunication*.

```
private void initCommunication() throws Exception {
      // service IDSProxy
      WSIDSProxy service;
      // URI WSDL
      String wsdlURI = config.getAddressWsdl();
      // URI Service
      String serviceURI = config.getAddressService();

      // lookup service
      ServiceClient serviceClient = ServiceClient.create(wsdlURI);
      serviceClient.setServiceURL(serviceURI);
      // get instance the IDSProxy for use the communication
      service = (WSIDSProxy)serviceClient.createProxy(WSIDSProxy.class);
      this.serviceCommunication = service;

}
```

Lista de Código 5.3: Trecho do método *initCommunication*

5.2.3 Implementação do IDS-Proxy

O IDS-Proxy é um componente que tem como finalidades principais:

- Receber os dados de um usuário final que foram enviados através do IDS-Client;
- Realizar a validação e formatação inicial dos dados;
- Enviar os dados obtidos ao IDS-NIDIA Remoto;
- Receber informações do IDS-NIDIA Remoto e encaminhá-las ao IDS-Client.

Um protótipo do IDS-Proxy foi desenvolvido para demonstrar a utilização das suas principais funcionalidades. Conforme explicitado no capítulo 4, o IDS-Proxy é desenvolvido como um *Web service*, devido à necessidade do modelo de IDS-NIDIA Remoto obter informações dos usuários finais através da Internet. O IDS-NIDIA Remoto usa agentes para executar suas principais funcionalidades, assim, fez-se necessário também

à criação de um agente que "traduz" os dados obtidos pelo IDS-Proxy (mensagens SOAP) em dados compreensíveis (mensagens ACL) para os agentes do IDS-NIDIA Remoto. O AgentProxy é responsável por fazer o link e a tradução dos dados do IDS-Proxy aos agentes do IDS-NIDIA Remoto.

Na Figura 5.7, apresentamos o diagrama de classe em UML do protótipo do IDS-Proxy.

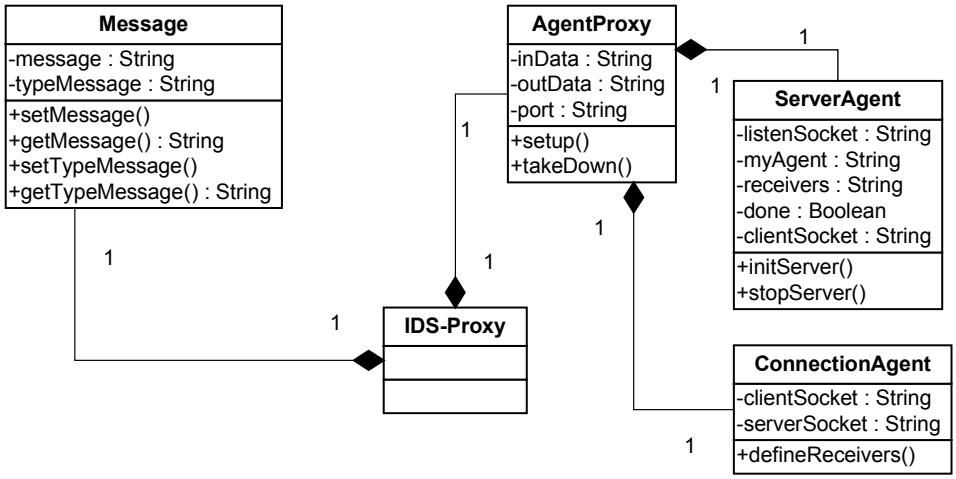

Figura 5.7: Diagrama de Classes do Protótipo IDS-Proxy (fragmento)

O desenvolvimento do protótipo do IDS-Proxy foi dividido em duas partes:

- Desenvolvimento do *Web service*;
- Desenvolvimento do AgentProxy.

O desenvolvimento do *Web service* do IDS-Proxy foi feito utilizando-se a ferramenta *Systinet Developer for Eclipse* versão 6.0. Fez-se uso de quase todas as funcionalidades desta ferramenta. Dentre as funcionalidades usadas, temos a criação de um *Web service* a partir de uma classe Java existente. Esta ferramenta permite a partir de uma classe Java, nos padrões de um EJB (*Enterprise Java Beans*), com métodos *get* e *set* públicos gerar uma outra classe que disponibiliza estes métodos como serviços de um *Web service*. Assim criou-se um *Web service* através da classe *Message*. Esta geração não é completamente automática, algumas modificações no código-fonte são necessárias para utilização do *Web service* gerado. O trecho de código na Lista de Código 5.4 apresenta a classe *Message* como um *Web Service*.

```
public class Message {
        // message in and out for the final users
        private String message =null;
        // type of message
        private String type = null;
        // host for the send message, used for the socket
        private String host=null;
        // port for the send message, used for the socket
        private int port=6789;
        ...
        // set message for the send ou receive
        public void setMessage(String msg){
                this.message = msg;
        }
        // set type for send or receive
        public void setType(String tp){
                this.type = tp;
        }
        // get the message
        public String getMessage(){
                return this.message;
        }
        // get Type for Message
        public String getType(){
                return this.type;
        }
        public void sendMessage(String sdmsg){
                Socket echoSocket = null;
                PrintWriter out = null;
                BufferedReader in = null;
        ...
        }
}
```

Lista de Código 5.4: Trecho da classe *Message* (fragmento)

O desenvolvimento do AgentProxy do IDS-Proxy foi feito utilizando-se a plataforma de desenvolvimento de agentes JADE versão 3.2. O AgenteProxy é baseado no código-fonte do SocketProxyAgent. O SocketProxyAgent é um agente desenvolvido pela TILAB (JADE), com o objetivo de permitir que informações externas possam chegar até a plataforma de um agente. Para isto o SocketProxyAgent é composto de dois *sockets* (server e client). A principal modificação realizada no código-fonte deste agente foi a inclusão de classes que permitissem o recebimento de mensagens SOAP e a conversão destas em mensagens ACL.

O projeto do AgentProxy está especificado da seguinte maneira:

- **Pacote br.nidia.agentproxy;**
- **Pacote br.nidia.agentproxy.ontology.**

O pacote **br.nidia.agentproxy** contém:

- A classe AgentProxy responsável por toda a tarefa de "tradução" e encaminhamento das mensagens recebidas pelo IDS-Proxy e que devem ser repassadas ao IDS-NIDIA Remoto. As classes ServerAgent e ConnectionAgent estão aninhadas a classe AgentProxy;
- O arquivo de recursos AgentProxy.xml utilizado pelo agente para permitir controlar a sua execução.

O pacote **br.nidia.agentproxy.ontology** contém:

- A classe *Message* responsável por montar o predicado usado pela ontologia MessageOntology. Esta classe contém o conteúdo dos dados provenientes de uma conexão capturada pelo IDS-Client e enviadas ao IDS-Proxy;
- A classe *MessageOntology* é responsável pela ontologia e é usada na comunicação entre o AgenteProxy e o IDS-Proxy e do AgentProxy com o IDS-NIDIA Remoto.

O trecho de código na Lista de Código 5.5 apresenta uma parte da classe ConnectionAgent aninhada a classe AgentProxy.

```
/**
 * Method run for the Class ConnectionAgent
 * Used for get the Message (EJB) and  set Msg (ACL)
 * Define Sender, Receiver, Language, Ontology and Content
 */
public void run()
{
    try {
    .
    .
    .
    done = false;
             while (!done)
             {
             // create the Message ACL
             msgACL = new ACLMessage(ACLMessage.INFORM);
             // set Agent Sender for the Message ACL
             msgACL.setSender(getAID());
             // set Agents Receivers for the Message ACL
             msgACL.addReceiver(new AID("AgentDestino@PC-MAURO:1099/JADE", AID.ISGUID));
             // set Language fro the Message ACL
             msgACL.setLanguage(msgSOAP.getLanguage());
             // set Ontology for the Message ACL
             msgACL.setOntology(msgSOAP.getOntology());
             // set the Content for teh Message ACL
             msgACL.setContent(msgSOAP.getContent);
             // send the Message ACL
             myAgent.send(msgACL);
    .
    .
    .
             } // END while (!done)
    }
    catch (Throwable any) {
        close(any);
        return;
    }
} // END run()
```

Lista de Código 5.5: Trecho da classe *ConnectionAgent* (fragmento)

5.2.4 Implementação dos NIDIA *Profiles*

A implementação dos NIDIA *Profiles* pode ser dividida em duas etapas.

- Desenvolvimento dos metamodelos usados para criação do repositório de metadados;
- Desenvolvimento das rotinas de manipulação do repositório de metadados.

No capitulo 4, todo o processo de criação do repositório de metadados foi apresentado. O processo de criação dos metamodelos e a importação dos metamodelos para o MDR *Browser for NetBeans IDE 3*.5.1 foram apresentados, finalizando com a geração das interfaces de manipulação dos elementos pertencentes ao metamodelo que são armazenados no repositório de metadados.

Aqui, nós apresentamos os procedimentos da segunda etapa da implementação dos NIDIA *Profiles*, ou seja, as rotinas de manipulação do repositório de metadados.

O protótipo responsável pela manipulação dos metadados chama-se UseDBAux. Na Figura 5.8, apresentamos o diagrama de classe em UML do protótipo do UseDBAux. Este protótipo foi escrito usando a linguagem de programação Java versão 1.4.2 e um conjunto de bibliotecas (arquivos JAR) distribuídos em um pacote denominado MDRStandalone. No MDRStandalone, encontram-se as bibliotecas responsáveis por dar suporte ao uso do MOF (*mof.jar*), JMI (*jmi.jar* e *jmiutils.jar*) e do próprio MDR (*mdrapi.jar*).

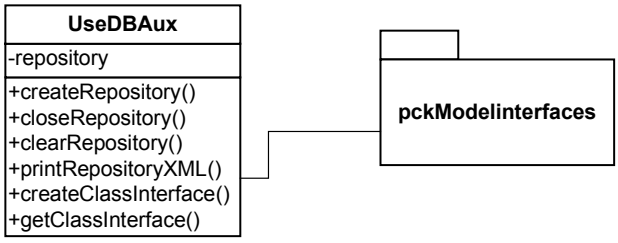

Figura 5.8: Diagrama de Classes do Protótipo *UseDBAux* (fragmento)

O projeto UseDBAux está dividido da seguinte forma:

- **Pacote br.nidia.usedbaux**

Este pacote contém a classe Usedbaux responsável por manipular o repositório de metadados;

- **Pacote br.nidia.usedbaux.pckmodelinterfaces**

Este pacote contém todas as interfaces geradas no processo de criação do repositório. As interfaces de instância e interfaces *factory* fazem parte deste pacote. Conforme explicitado no capítulo 4, quando da criação do repositório, duas interfaces são geradas para cada elemento do metamodelo (classes e associações): as interfaces de instância e interfaces *factory*;

- **Pasta resources**

A pasta *resources* contém o metamodelo no formato XMI conforme ao MOF utilizado na rotina de montagem e abertura do repositório de metadados.

Class UseDBAux

A classe UseDbAux é responsável pelas seguintes funcionalidades:

- Montar o repositório de metadados: a montagem do repositório é feita através da leitura do arquivo XMI (conforme a MOF) que contém os elementos (classes e associações) do metamodelo a ser carregado no repositório. Assim, a rotina de montagem lê o arquivo e cria uma estrutura persistente no repositório MDR com as classes e associações a serem manipuladas;
- Abrir o repositório de metadados: a abertura é feita obtendo-se uma instância do repositório montado. Para abrir o repositório, é necessária que seja informada a localização do repositório montado anteriormente;
- Fechar o repositório: antes de finalizar a utilização do repositório, é necessário que este seja fechado, para evitar consumo de recursos no sistema. A classe executa uma rotina de *shutdown* para fechar o repositório de forma correta.

O trecho de código na Lista de Código 5.6 apresenta uma parte da classe UseDBAux.

```
public class ExecuteDBAux {
        // name of a MOF extent that will serve as a target extent for the UML2MOF transformation
        private static final String MY_INSTANCE = "pckDBAux_Inst";
        // name of a UML extent (instance of UML metamodel) that the UML models will be loaded into
        private static final String My_MM = "pckDBAux";
        // repository
        private static MDRepository rep;
        // XMI reader
        private static XmiReader reader;
        // XMI writer
        private static XmiWriter writer;
        public static void main(String[] args) {
        try {
                System.setProperty("org.netbeans.mdr.persistence.Dir","C:/modelNidia/mdr/mdr");
                // look up an implementation of XmiReader interface
                reader = (XmiReader) Lookup.getDefault().lookup(XmiReader.class);
                // look up an implementation of XmiWriter interface
                XmiWriter writer = (XmiWriter) Lookup.getDefault().lookup(XmiWriter.class);
                // open file for create XML
                FileOutputStream out = new FileOutputStream("C:/modelNidia/xml/DBAux.xml");
                // connect to the repository
                rep = MDRManager.getDefault().getDefaultRepository();
                .
                .
                .
        }catch (Exception e) {
                ErrorManager.getDefault().notify(ErrorManager.ERROR, e);
        }
        // close repository
        public void CloseRepository(){
            rep.shutdown();
      }
       // load XMI in repository
        private static MofPackage getUmlPackage() throws Exception {
           // get the MOF extent containing definition of UML metamodel
           ModelPackage thisMM = (ModelPackage) rep.getExtent(My_MM);
           if (thisMM == null) {
             // it is not present -> create it
             thisMM = (ModelPackage) rep.createExtent(My_MM);
           }
          // find package named "UML" in this extent
          MofPackage result = getUmlPackage(thisMM);
          if (result == null) {
```

```
            // it cannot be found -> UML metamodel is not loaded -> load it from XMI
            String resource =
PckDbauxPackage.class.getResource("/resources/MetaModelDBAuxDI.xmi").toString();
          reader.read(resource, thisMM);
         // try to find the "UML" package again
          result = getUmlPackage(thisMM);
          }
          return result;
      }
}
```

Lista de Código 5.6: Trecho da classe *UseDBAux* (fragmento)

5.2.5 Atualização dos Agentes NIDIA

Conforme explicitado no capítulo 4, o IDS-NIDIA teve seus agentes atualizados da plataforma ZEUS para a plataforma JADE. O projeto está dividido em sete pacotes, duas pastas de recursos e um conjunto de bibliotecas de suporte. A Figura 5.9 apresenta o Diagrama de Pacotes em UML do projeto do IDS-NIDIA Remoto. Este diagrama exibe a relação de dependência entre os pacotes do IDS-NIDIA Remoto.

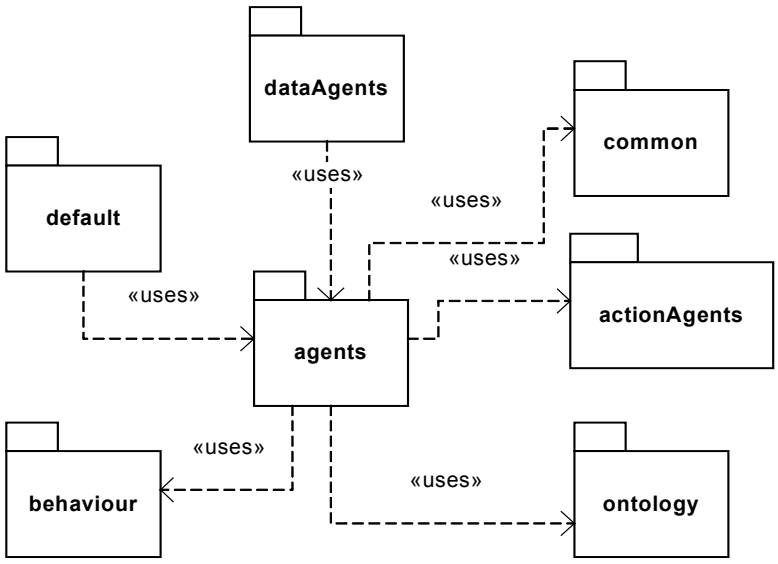

Figura 5.9: Diagrama de Pacotes em UML do IDS-NIDIA (fragmento)

Segue a lista de cada um destes pacotes:

- Pacote **default;**
- Pacote **actionAgents;**
- Pacote **agents;**
- Pacote **behaviour;**
- Pacote **common;**
- Pacote **dataAgents;**
- Pacote **ontology.**

O **pacote default** contém as classes responsáveis por:

- Iniciar o NIDIA;
- Dar suporte a implementação dos comportamentos dos Agentes;
- Pela manipulação da rede neural de auxílio à detecção de intrusos.

O **pacote actionAgents** contém as classes responsáveis por:

- Criar o *log* das conexões realizadas com os agentes;
- Gerar a mensagem de envio das conexões capturadas;
- Realizar a notificação das Contramedidas.

O **pacote agents** contém as classes responsáveis por:

- Fornecer uma interface gráfica do IDS;
- Inicializar os Agentes do sistema e inicializar os recursos necessários tais como: o arquivo de configuração agents.txt (responsável por indicar onde os agentes serão inicializados) e o repositório de metadados.

O **pacote behaviour** contém as classes responsáveis pelos comportamentos dos agentes. Os comportamentos definem as atividades que o agente deve executar para que possa alcançar os objetivos a eles designado.

Estes comportamentos são os seguintes:

- Envio e recebimento de uma mensagem de PING para indicar se o agente ainda está respondendo;
- Realizar a associação dos dados recebidos pelo Agente MCA e então usar de uma rede neural, usando o algoritmo de associação binária para tratar o incidente;
- Iniciar os agentes de base que compõe o sistema;
- Controlar as execuções dos agentes;
- Capturar os pacotes utilizados para análise;
- Capturar as capacidades dos sistemas em que os agentes estão executando.

O **pacote common** contém as classes responsáveis:

- Pela captura e manipulação das informações das máquinas que interagem com o IDS-NIDIA Remoto;
- Pela indicação dos serviços que o IDS-NIDIA Remoto dispõe para os usuários;
- Pela checagem da comunicação entre os agentes do sistema.

O **pacote dataAgents** possui a classe responsável pela manipulação do repositório de metadados. Esta classe realiza:

- A inicialização e se necessário criação do repositório de metadados;
- Recuperação das informações armazenadas no repositório de metadados;
- Pelo fechamento e exclusão do repositório de metadados.

O **pacote ontology** contém as classes responsáveis pela ontologia utilizada entre os agentes do IDS-NIDIA Remoto. Sendo assim, estas classes devem:

- Definir as informações que são enviadas com os dados provenientes de uma conexão capturada;
- Definir as informações que são enviadas com os dados provenientes de uma conexão capturada.

As duas pastas de recursos são as pastas ***images*** e ***resources***. A pasta *images* contém as imagens que são utilizadas no sistema, e a pasta *resources* possui arquivos de configuração do IDS-NIDIA Remoto (agents.txt, arqrede.txt,back2.txt,chaves.txt e prdb.txt).

As bibliotecas utilizadas para suporte na execução do sistema são:

- Bibliotecas do JADE;
- Bibliotecas do pacote MDRStandalone;
- Biblioteca JPCAP.

5.3 Execução e Resultados obtidos

Esta seção apresenta:

- O cenário para execução dos protótipos;
- A execução dos protótipos implementados;
- Os resultados obtidos através da execução dos protótipos.

Os resultados são obtidos através da simulação de um ataque ao sistema do usuário final. O ataque sob o qual o sistema do usuário final foi submetido é denominado técnica de adivinhação de senha [34].

5.3.1 Cenário de Execução

O cenário da Figura 5.10 foi montado para demonstrar a execução do protótipo e para ser usado na simulação do ataque. O cenário foi montado com auxílio do laboratório de informática da Pós-Graduação da UFMA.

Este cenário é composto por:

- Um usuário conectado à Internet, sem ferramentas de segurança instaladas em seu sistema. Este elemento do cenário é chamado de Domínio do Usuário (*User domain*);
- Uma máquina com capacidade servidora em que o *Web service* do IDS-Proxy, o IDSProxyWS, em conjunto com seu agente JADE, o AgentProxy e o Agente de Segurança Local do IDS-NIDIA Remoto estejam instalados e em execução;
- Um conjunto de máquinas dispostas em rede onde os agentes do IDS-NIDIA Remoto estejam executando. Este elemento do cenário é chamado de Domínio do IDS-NIDIA Remoto (Remote NIDIA *domain*).

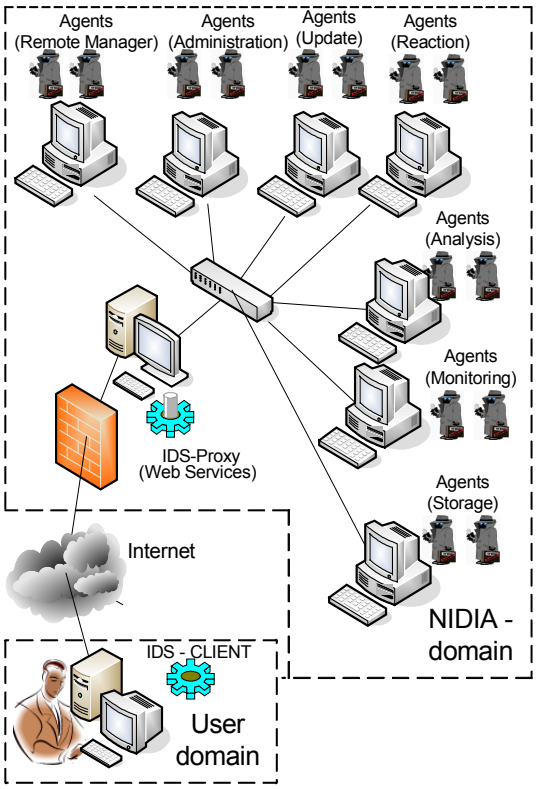

Figura 5.10: Cenário de utilização do protótipo do modelo

O cenário pode ser descrito da seguinte forma: um usuário final, que está conectado à Internet e que não possui um IDS local, solicita os serviços do IDS-NIDIA Remoto. Esta solicitação parte do IDS-Client e através do IDS-Proxy é aceita pelo IDS-NIDIA Remoto. Após o aceite, o IDS-Client é ativado no sistema do usuário final e então inicia seu processo de captura de informações. Os dados são enviados ao IDS-Proxy que ao recebê-los, encaminha-os ao IDS-NIDIA Remoto para análise. Estando estes procedimentos em

execução, a simulação de um ataque de adivinhação de senhas é realizado no sistema do usuário final. Desta forma, o cenário contribui na obtenção de resultados, visto que o protótipo detecta o ataque provendo contramedidas tais como a desabilitação da conexão com a fonte de ataque (*host* malicioso) ou redirecionamento para um *honeypot*.

Mas para fazer uso do cenário é necessário que os protótipos estejam em execução. O processo de execução dos protótipos para demonstração do uso do IDS-NIDIA Remoto, é feito obedecendo os seguintes passos:

1. Inicialização e execução dos agentes do IDS-NIDIA Remoto;
2. Inicialização e execução do Web serivice e do AgenteProxy do IDS-Proxy;
3. Inicialização e execução do IDS-Client.

5.3.2 Execução do protótipo

Demonstramos nesta seção a execução de cada protótipo pertencente ao modelo proposto. A forma de inicialização, o uso e a aplicabilidade de cada protótipo do cenário apresentado na Figura 5.10 são descritos nas próximas seções.

Execução do IDS-NIDIA Remoto

Para criar o NIDIA *domain*, é necessário que o IDS-NIDIA Remoto seja iniciado, a partir de um *host* habilitado a montar uma plataforma de agentes JADE. Para a montagem do NIDIA *domain* foram utilizados três *hosts*, cujos IPs são: 192.168.2.24, 192.168.2.25, 192.168.2.26. A plataforma JADE foi distribuída entre esses *hosts*. O *Main-Container* está localizado no *host* 192.168.2.26 e os demais *containers* (*containers* comuns) ficaram hospedados nos demais *hosts*. A Figura 5.11, apresenta o Gerenciador Remoto de Agentes do JADE, o RMA (*Remote Management Agent*), que destaca a plataforma de agentes JADE montada para o IDS-NIDIA Remoto.

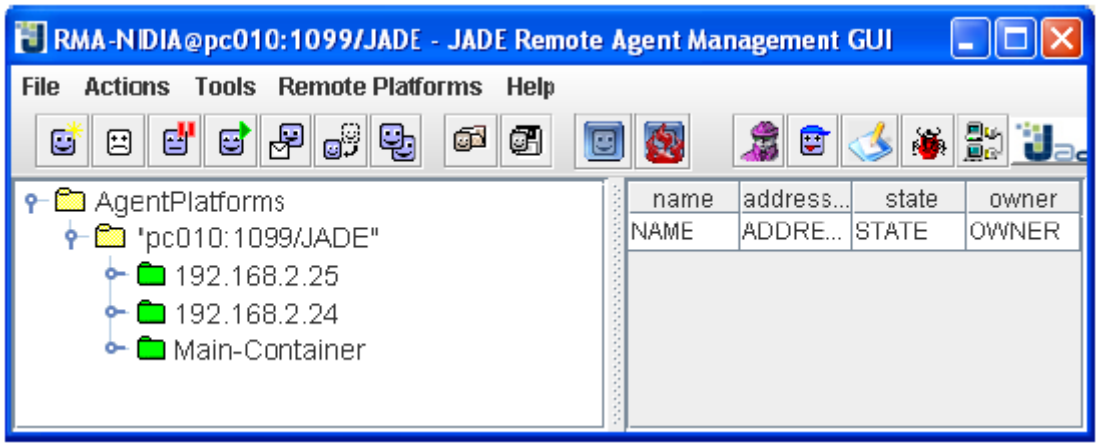

Figura 5.11: Plataforma de Agentes JADE do IDS-NIDIA

O IDS-NIDIA Remoto é iniciado através da classe StartNIDIA. Esta classe configura a plataforma de agentes JADE que será usada pelo IDS-NIDIA Remoto. A classe StartNIDIA tem em seu método *main* a criação do *Main-Container*, e a criação e inicialização do Agente RMA-NIDIA e do Agente LSIA. O trecho de código na Lista de Código 5.7 apresenta um fragmento da classe StartNIDIA.

```
/**
 * This class is used to begin IDS NIDIA.
 */
public class StartNIDIA {

  public static void main(String args[]) {

    try {
      Runtime rt = Runtime.instance();
      rt.setCloseVM(true);
      Profile pMain = new ProfileImpl(null, 1099, null);
      // create main-container
      AgentContainer mc = rt.createMainContainer(pMain);
      // create Agent RMA
      AgentController rma = mc.createNewAgent("RMA-NIDIA", "jade.tools.rma.rma", new Object[0]);
      // start Agent RMA
      rma.start();
     // create Agent LSIA
     AgentController agentstart = mc.createNewAgent("LSIA", "agents.LSIAAgent", args );
     // start Agent LSIA
     agentstart.start();
     }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Lista de Código 5.7: Trecho da classe *StartNIDIA* (fragmento)

O Agente LSIA, através do seu comportamento LSIABehaviourAddAgent, é o agente que inicia os demais agentes do IDS-NIDIA Remoto. Isto é feito através da leitura do arquivo de recurso *agents.txt*. O arquivo de recursos *agents.txt* contém:

- Os *hosts* que serão utilizados pela plataforma de agentes;
- Os agentes que serão usados pelo IDS-NIDIA Remoto;
- O *host* em que cada agente será iniciado.

Para a demonstração deste protótipo as informações do arquivo *agents.txt* foi utilizado de forma estática.

Após inicialização, o Agente LSIA fica executando no *Main-Container*. Os demais agentes do IDS-NIDIA Remoto também são iniciados (no *Main-Container* e nos c*ontainers* comuns) e, consequentemente, começam a executar seus comportamentos.

A Figura 5.12 apresenta os agentes do IDS-NIDIA Remoto nos *containers* comuns.

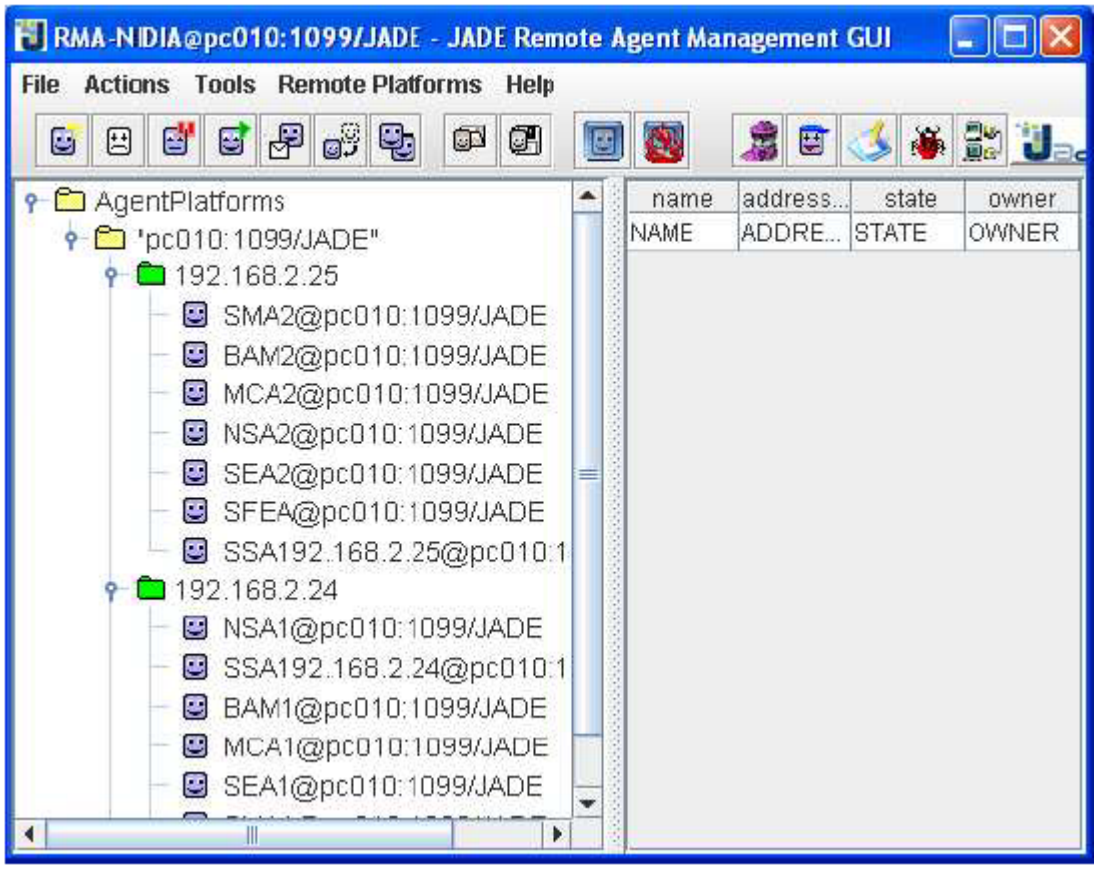

Figura 5.12: Agentes do IDS-NIDIA Remoto nos *containers* comuns

A Figura 5.13 mostra os agentes do IDS-NIDIA Remoto no *Main-Container*.

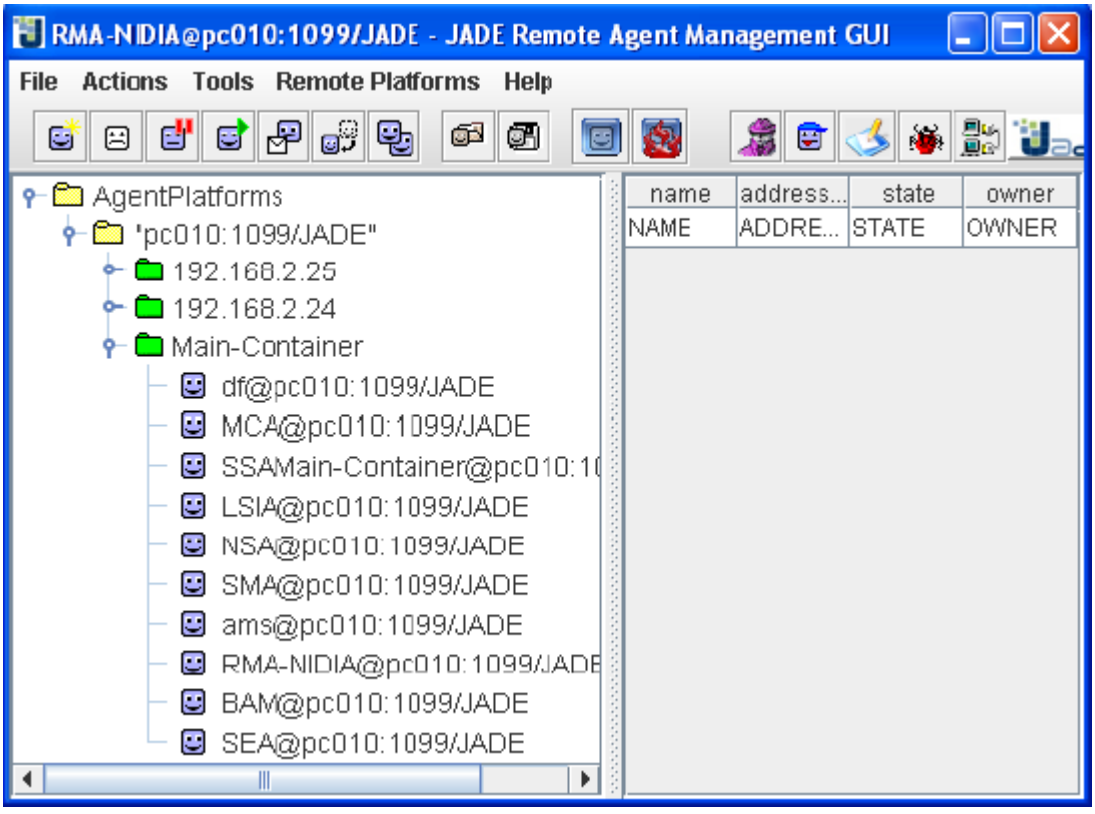

Figura 5.13: Agentes do IDS-NIDIA no *Main-Container*

Após inicialização do IDS-NIDIA Remoto, o NIDIA *domain* está configurado. O IDS-NIDIA Remoto está pronto para iniciar suas atividades de detecção através do comportamento de seus agentes.

Execução do IDS-Proxy

A execução do IDS-Proxy é iniciada através da inicialização do *Systinet Server for* Java. Após a inicialização do *Systinet Server for* Java, o IDSProxyWS, representando o *Web service* do IDS-Proxy, fica disponível para ser usado pelo IDS-Client e pelo IDS-NIDIA. A Figura 5.14 apresenta o *Systinet Server for* Java iniciado com sua árvore de serviços disponíveis. Um dos nós da árvore de serviços é o IDSProxyWS.

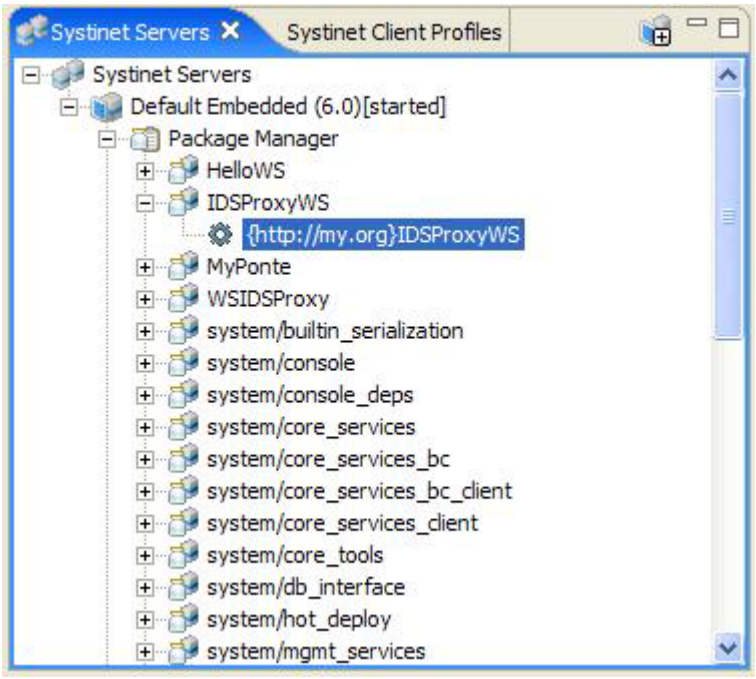

Figura 5.14: *Systinet Server for* Java com o IDSProxyWS

O IDSProxyWS é um *Web service* criado com auxílio da ferramenta *Systinet Developer for Eclipse* versão 6.0. Um dos principais serviços disponibilizados pelo IDSProxyWS é o recebimento dos dados capturados pelo IDS-Client e posteriormente o envio destes dados para o IDS-NIDIA Remoto. A segunda parte do serviço disponibilizado pelo IDSProxyWS é feito com auxílio do Agentproxy. O envio dos dados ao IDS-NIDIA Remoto é feito através da abertura de *sockets* (*client* e *server*) responsáveis pela comunicação com o AgentProxy. O trecho de código na Lista de Código 5.8 apresenta um

fragmento do método *sendMessage* responsável por enviar os dados recebidos dos usuários finais ao AgentProxy.

```
/**
 * This class is used for the IDSProxyWS.
 */
public class Message {
        // message in and out for the final users
        private String message =null;
        // type of message
        private String type = null;
        // host for the send message, used for the socket
        private String host=null;
        // port for the send message, used for the socket
        private int port=6789;
        ...
        public void sendMessage(String sdmsg){
                Socket echoSocket = null;
                PrintWriter out = null;
                BufferedReader in = null;
                try {
                        echoSocket = new Socket(this.host, this.port);
                                out = new PrintWriter(echoSocket.getOutputStream(), true);
                                in = new BufferedReader(new InputStreamReader(
                                                                echoSocket.getInputStream()));
                }
                catch (UnknownHostException e) {
                                System.err.println("Don't know about host: " + host);
                                System.exit(1);
                 }
        ...
}
```

Lista de Código 5.8: Trecho do método *sendMessage* (fragmento)

O AgentProxy é um agente JADE responsável por receber os dados obtidos pelo IDSProxyWS, através do protocolo SOAP. Após o AgentProxy receber os dados provenientes do processo de captura das informações do usuário final, encaminha estes dados aos agentes do IDS-NIDIA Remoto para que possam ser analisados pela sua plataforma de agentes. O AgentProxy possui os métodos *setup*, *takedown* e *log*, sendo o método *setup* o principal. Possui ainda duas classes *Threads* aninhadas, a classe *ServerAgent* e a classe *Connection*. Os métodos, *takedown* e *log* são responsáveis respectivamente pelo final da execução e pela escrita do *log*. No método *setup* o AgentProxy ativa ou desativa a escrita em seu arquivo de *log* de execução (*logFile*), realiza a leitura de seu arquivo de configuração *AgentSocket.xml* para identificar em que porta o *ServerSocket* associado a ele deverá ficar escutando e quais os agentes que irão receber as mensagens que chegam até ele. No *AgentSocket.xml*, se tiver sido indicado que a porta a ser usada tem o valor 0, o sistema entende que será usado o valor *default* (6789), caso contrário ele ficará esperando um valor de porta, sendo ideal o uso de portas altas. Da mesma forma, se for indicado um * (asterisco) na lista de agentes que o AgentProxy se comunicará, este entende que serão todos os agentes. Caso contrário os agentes deverão ter seus nomes separados por espaço. Sendo que cada agente de plataforma diferente deve ter seu nome completo colocado dentro do arquivo (nickname@endereço:porta/JADE). Após a leitura do *AgentSocket.xml*, um vetor com todos os agentes que poderão interagir com AgentProxy é

montado e em seguida sua *Thread ServerAgent* é iniciada. A Figura 5.15 apresenta o AgentProxy executando no *Main-Container* em sua plataforma de agentes JADE.

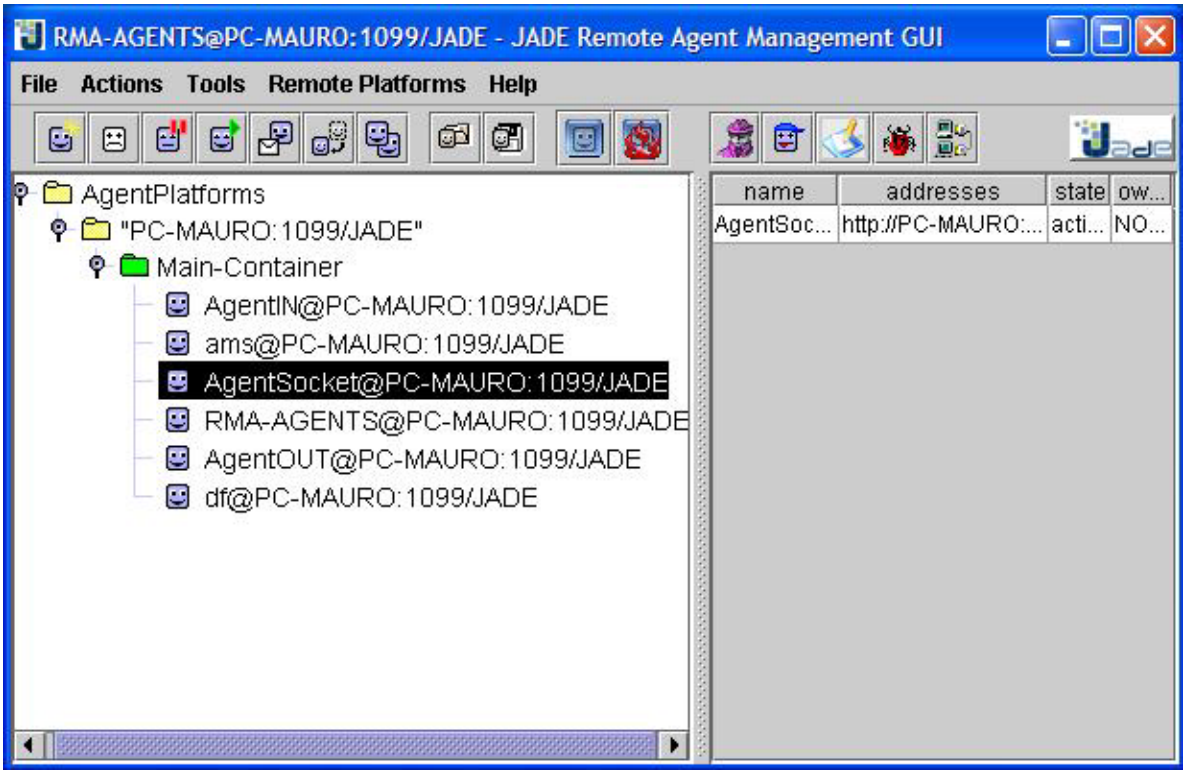

Figura 5.15: AgentProxy executando no *Main-Container*

Estando o serviço do *Web service* IDSProxyWS inicializado e a plataforma de agentes do AgentProxy também criada e em execução, o IDS-Proxy está ativado e pronto para receber informações dos usuários finais e enviar estes dados recebidos para IDS-NIDIA Remoto.

Execução do IDS-Client

O IDS-Client tem sua execução associada à configuração e montagem do *User domain*. Para ativar o protótipo do IDS-Client é necessário que este seja chamado a partir da linha de comando. Futuramente, o IDS-Client poderá ser colocado como serviço a ser executado na inicialização do sistema operacional do usuário final. O IDS-Client quando é iniciado, executa uma série de checagens e configurações, tais como: configuração e checagem da comunicação, captura de dados da performance do sistema, captura dos dados trafegados pelo sistema do usuário final, configuração das contramedidas e a sua área de

notificação. Todas estas atividades estão disponíveis para o usuário final através da sua barra de tarefas.

Para que o IDS-Client desempenhe suas atividades é necessário que o IDS-Proxy esteja com seus serviços ativados (IDSProxyWS e AgentProxy ativados). Desta forma, o IDS-Client, através do método *openConnection* da *facility Communication*, executa uma tentativa de conexão com o IDS-Proxy obtendo como resultado uma das duas mensagens:

- *Active Communication*: caso a conexão tenha sido realizada com sucesso, indicando que o IDS-proxy está ativo;
- *Communication Refuse*: caso esta conexão não possa ser estabelecida.

O IDS-Client faz uso da sua área de notificação para informar ao usuário se o IDS-Proxy está ou não ativo. Estando o IDS-Proxy ativo, o IDS-Client inicia o processo de levantamento dos dados de performance do sistema do usuário, através da *facility SystemDiagnostic*, e posteriormente inicia a captura dos dados trafegados no sistema do usuário final, através da *facility Capture*.

A Figura 5.16 apresenta a tela do IDS-Client no momento de sua inicialização, demonstrando principalmente sua execução inicial. Na Figura 5.16, podemos verificar na área de notificação do IDS-Client, que a conexão com o IDS-Proxy está ativa e que os dados de performance do sistema estão sendo capturados.

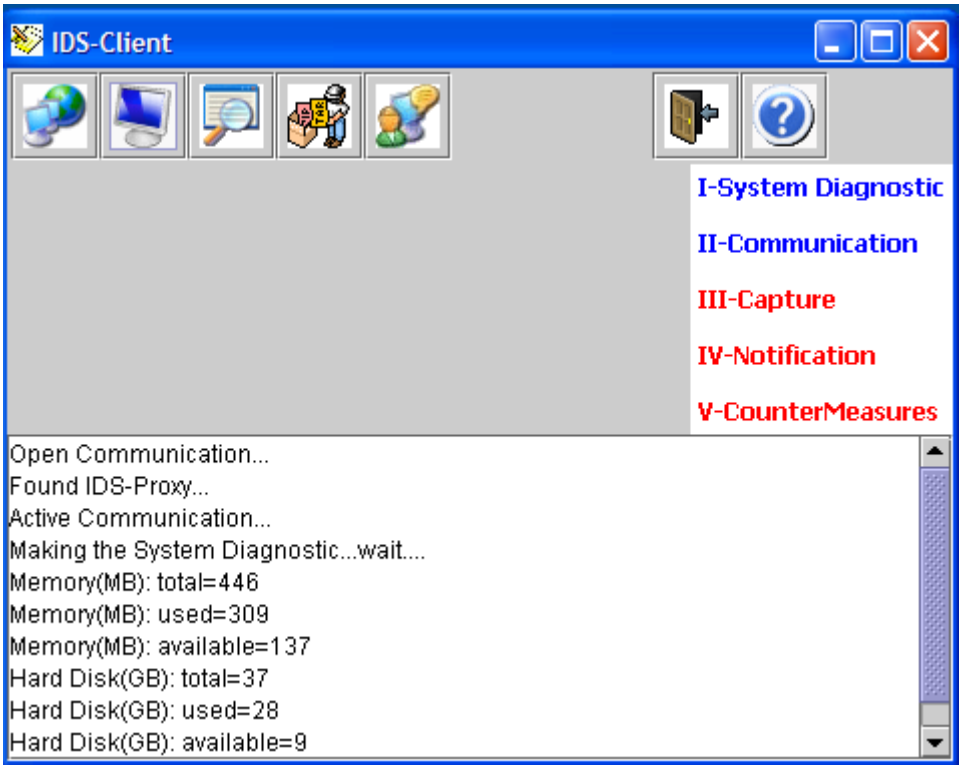

Figura 5.16: IDS-Client em execução

Desta forma, o IDS-Client inicia seu processo de captura e envio dos dados do sistema do usuário final para o IDS-Proxy. O IDS-Proxy ao receber estes dados, os encaminha para IDS-NIDIA Remoto, para que o IDS-NIDIA Remoto possa analisá-los. O IDS-Client fica em execução aguardando receber do IDS-NIDIA Remoto notificações de ataques e as contramedidas necessárias a tomar. Isto quando, os dados enviados pelo IDS-Client e analisados pelo IDS-NIDIA Remoto apontarem uma detecção de invasão.

5.3.3 Captura, Detecção e Contramedidas

Após a inicialização de todos os protótipos e consequentemente configuração do NIDIA *domain* e *User domain*, podemos então fazer a simulação de um ataque ao sistema do usuário final.

No processo de simulação, o sistema do usuário é submetido a um ataque de rede baseado em conteúdo. Os ataques de rede baseados em conteúdo fazem uso do campo de dados de um pacote TCP/IP (*payload*) [34]. O objetivo principal neste tipo de ataque é conseguir acesso privilegiado para capturar/destruir/adulterar informações importantes, implantar um software para atacar outras máquinas utilizando a vítima como um "zumbi" ou utilizar o poder de processamento desta máquina para atividades não autorizadas. Dentre as formas de se efetivar esses ataques utilizando o *payload*, escolhemos para uso do procedimento de simulação a técnica de adivinhação de senha. Existem diversos programas disponíveis na Internet, para executar este tipo de ataque. Entre os programas encontrados, podemos citar: o *Legion*, *CyberCop Scanner* e *NTInfoScan*. No nosso caso, apesar de estudarmos o uso destes programas, não utilizamos nenhum destes programas. Para nossa simulação de ataque fizemos a modificação de um pacote capturado do sistema do usuário final, incluindo neste, palavras-chave geralmente recorrentes neste tipo de ataque. Desta forma, o pacote modificado que partirá do usuário final chegará ao IDS-NIDIA Remoto para ser analisado. O IDS-NIDIA remoto através do seu processo de análise irá procurar no pacote recebido pela existência de palavras que identifiquem a ocorrência de um ataque.

Neste processo de simulação, o IDS-Client está em execução no sistema do usuário final. Sendo assim, os dados do sistema do usuário final estão sendo capturados e enviados ao IDS-Proxy. O IDS-Proxy recebe estes dados através do seu *Web service*, IDSProxyWS. A Figura 5.17 apresenta uma mensagem SOAP, capturada pelo IDSProxyWS, contendo

informações capturadas do sistema do usuário final. Esta mensagem foi capturada no momento em que o usuário acessava o site http://www.terra.com.br[19].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<e:Body>        <n0:setMessage xmlns:n0="http://systinet.com/wsdl/br/idsproxyws/pck/">          <n0:msg
i:type="d:string"><html lang="pt"><head><title>Terra - Qual &eacute; a sua?</title><meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1"><script type="text/javascript" language="javascript1.1"
src="http://www.terra.com.br/ads/capa/tag_comercial.js"></script><script type="text/javascript"
language=JavaScript><!—"Login incorrect"          </n0:msg>       </n0:setMessage>      </e:Body>
</e:Envelope>
```

Figura 5.17: Dados capturados no formato SOAP

A mensagem em formato SOAP recebida pelo IDSProxyWS é encaminhada ao AgentProxy. O IDSProxyWS encaminha esta mensagem por meio de um *socket* embutido em seu serviço *sendMessage*. A mensagem ao chegar no Agentproxy é então transformada em mensagem ACL. A Figura 5.18 apresenta uma mensagem ACL, transformada a partir da mensagem SOAP apresentada na Figura 5.17.

```
(INFORM
 :sender  ( agent-identifier :name AgentProxy@PC-MAURO:1099/JADE  :addresses (sequence
http://PC-MAURO:7778/acc ))
 :receiver  (set ( agent-identifier :name SMA1@192.168.2.25:1099/JADE ) )
 :content  "<html lang="pt"><head><title>Terra - Qual &eacute; a sua?</title><meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1"><script type="text/javascript" language="javascript1.1"
src="http://www.terra.com.br/ads/capa/tag_comercial.js"></script><script type="text/javascript"
language=JavaScript><!—Login incorrect"  :language  fipa-sl  :ontology  Messagem-ontology )
```

Figura 5.18: Dados capturados no formato ACL

A mensagem transformada em formato ACL pelo AgentProxy é encaminhada ao agente SMA do IDS-NIDIA Remoto.

O agente SMA é responsável pela formatação e pela checagem inicial dos dados capturados:

- Da rede local por meio do agente *NetSensor*;
- Do usuário final por meio do IDS-Client.

Isto é realizado através do comportamento SMABehaviour. O comportamento SMABehaviour verifica o número de ocorrências das palavras-chave (armazenadas no arquivo de recurso chaves.txt) dentro da mensagem recebida. Esta verificação é feita através de expressões regulares. A contagem do número de palavras é enviada para o agente

[19] Esta mensagem foi obtida com auxílio da ferramenta *SOAPSpy* da *Systinet.*

SEA. O trecho de código na Lista de Código 5.9 apresenta um fragmento do método *action* do comportamento SMABehaviour.

```
/**
 public void action () {
        . . .
        // get state for Agent
       myAgent.getState();
       // receive the message
        ACLMessage msg = myAgent.receive (mt);
       . . .
       try {
              enum_chaves = tabela_chaves.keys();
              //read the message and count keywords found in the message
              while (enum_chaves.hasMoreElements()) {
                     chave = (String) enum_chaves.nextElement();
                     p = Pattern.compile(chave);
                     mensagem.setDadosStr (mensagem.getDadosStr());
                     m = p.matcher (mensagem.getData());
                     while (m.find()) {
                          cont_chaves = (Integer) tabela_chaves.get(chave);
                          tabela_chaves.put(chave, new Integer(cont_chaves.intValue() + 1));
                     }
              }
              enum_chaves = tabela_chaves.keys();
              //write result in file arq_count
              try {
                     dados_arqcont.writeBytes(resultado);
                     dados_arqcont.writeByte(13);
                     dados_arqcont.writeByte(10);
        . . .
}
```

Lista de Código 5.9: Trecho do método *action* – *SMABehaviuor* (fragmento)

De posse deste conteúdo, o agente SEA provê o treinamento da rede neural usando o algoritmo de *BackPropagation* para gerar o vetor de contagem de palavras-chave e o grau de severidade do conteúdo. Estes dados e o identificador da conexão são enviados ao agente MCA. O agente MCA é responsável pela tarefa de administrar as vulnerabilidades encontradas durante o processo de análise.

Os dados capturados e analisados pelo agente SEA quando recebidos pelo agente MCA são encaminhados para o agente BAM para que este possa identificar a real ocorrência de um ataque.

O agente BAM identifica uma possível vulnerabilidade no sistema. O agente BAM através do comportamento BAMBehaviour recebe os dados enviados pelo agente MCA e aplica um filtro nos dados recebidos para identificar a similaridade com as informações armazenadas na base IIDB. Através da análise comparativa entre os dados recebidos e as informações sobre padrões de intrusão armazenada no IIDB, o agente BAM informa ao agente MCA se a conexão de onde os dados foram obtidos é de um atacante ou não. O trecho de código na Lista de Código 5.10 apresenta um fragmento do método *action* do comportamento BAMBehaviour.

```
public void action () {
   ACLMessage msg = myAgent.receive (mt);
   String Resultado;
   if (msg != null) {
   try {
      ContentElement ce = manager.extractContent (msg);
      String remetente = null;
      if (ce instanceof Message) {
         Message mensagem = (Message) ce;
         remetente = mensagem.getagenteOrigem();
         if (!(remetente.indexOf("MCA") < 0)) {
            String palavrasChave = mensagem.getEntrada ();
            Filtrador Filtra = new Filtrador();
            Resultado = Filtra.recebeConexaoSuspeita(palavrasChave);
            // send data go back for the MCA
            mensagem.setNomeAtaque (Resultado);
            mensagem.setagenteOrigem("BAM");
            this.envia_dados (mensagem, "MCA");
        }
     }
  } catch (Exception e) {
      System.out.println("Error!");
 }
...
}
```

Lista de Código 5.10: Trecho do método *action* - *BAMBehaviuor* (fragmento)

O Agente BAM ao detectar que o ataque ocorreu retorna ao agente MCA. O agente MCA, então, consulta na base de estratégias e obtém as contramedidas apropriadas a serem tomadas. A notificação e as contramedidas são enviadas para o LSIA e para o AgentProxy. O LSIA avisa o Administrador do Sistema NIDIA e AgentProxy encaminha a informação para o IDSProxyWS para que o IDS-Client possa ser notificado e receber as contramedidas.

A Figura 5.19 apresenta a notificação recebida pelo IDS-Client.

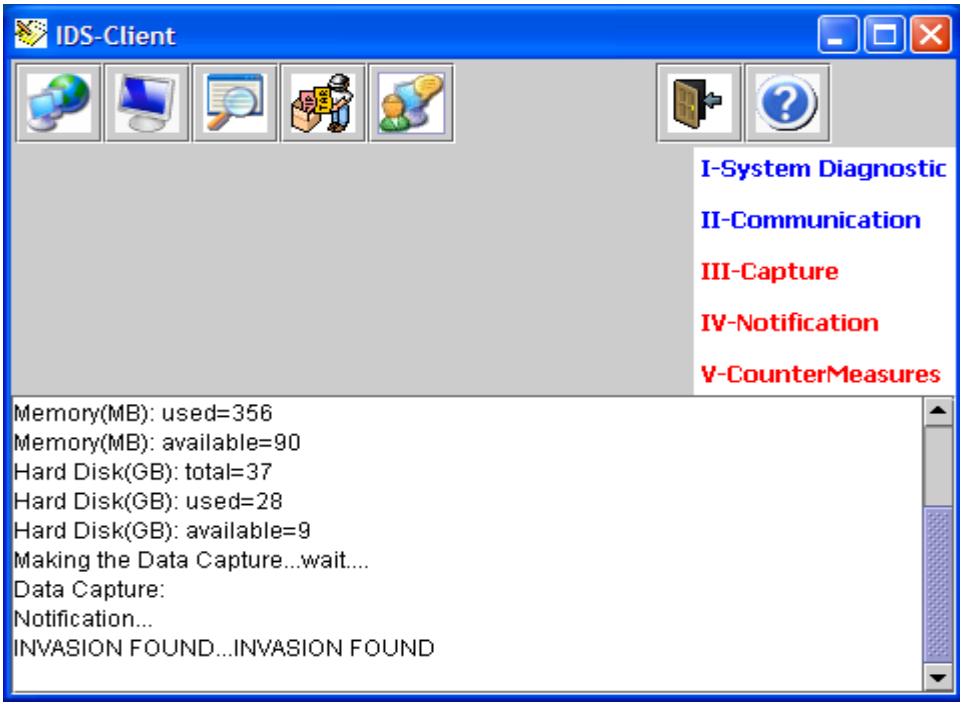

Figura 5.19: Notificação de uma invasão

A Figura 5.19 apresenta o resultado da captura do pacote formatado por nós no processo de simulação de ataque. Na nossa simulação, indicamos na mensagem enviada ao IDS-NIDIA Remoto, o IP da maquina que estava tentando descobrir a senha do usuário final. Este IP é usado na Contramedida a ser tomada pelo IDS-Client. Uma das contramedidas armazenadas no banco de estratégias é encerrar a comunicação com este IP. Esta contramedida é tomada pelo IDS-Client, inibindo todos os pacotes que vem deste IP suspeito. A *facility* Countermeasures executa esta ação e recusa todos os pacotes que são oriundos deste IP.

5.3.4 Resultados obtidos

O principal resultado obtido das etapas de desenvolvimento dos protótipos foi a demonstração da viabilidade do sistema. Os protótipos merecem atenção especial principalmente em fatores como segurança da ferramenta IDS-Client, disponibilidade e escalabilidade do IDS-Proxy e do IDS-NIDIA. Fatores não incluídos no estudo e nem no desenvolvimento das ferramentas. Alguns pontos importantes são percebidos:

- É permitido o uso do IDS-Client em múltiplas plataformas, pois seu desenvolvimento foi baseado em tecnologias que objetivam a interoperabilidade;
- O usuário final irá fazer uso de sistema IDS cuja base de assinaturas é dinâmica tendo uma maior capacidade de detecção e consequentemente, proteção;
- O usuário final poderá fazer uso do IDS-NIDIA Remoto sem necessidade de grandes instalações em seu sistema, devido a sua característica de distribuição em pacotes *jars*;
- O IDS-Client pode ser considerado um “cliente magro”, exigindo pouca capacidade de processamento do sistema do usuário final, pois as atividades que exigem grande capacidade de processamento (serviços de detecção) são executadas pelo IDS-NIDIA Remoto;
- O IDS-NIDIA Remoto pode sofrer atualizações em seus agentes e em suas bases, sem prejudicar o processamento do sistema dos usuários finais. Os usuários farão uso de um sistema atualizado e mais robusto.

É pertinente e muito claro que ainda há muito para desenvolver e aprimorar. No entanto, esboçamos aqui a capacidade de desenvolvimento de uma ferramenta de segurança essencial aos usuários finais e que, devido às tecnologias utilizadas, podem ser integradas

com demais ferramentas tais como *firewall*, antivírus e anti-*spywares*. Apontando desta forma, para o desenvolvimento de uma suíte de ferramentas (*firewall*, antivírus e anti-*spywares*) remotas, integradas e inteligentes.

6 Conclusões e Trabalhos Futuros

Este capítulo discute sobre as contribuições deste trabalho, as considerações finais sobre os resultados alcançados, as limitações de nosso trabalho e as sugestões para trabalhos futuros.

6.1 Contribuições do Trabalho

Existe atualmente uma conscientização por parte da comunidade das tecnologias da informação de que a segurança da informação é um fator fundamental em todos os ambientes que fazem uso dos dados digitais. Tal afirmativa tem fundamentação na crescente preocupação das organizações em proteger seus dados de invasores internos e externos. Além das organizações, algumas iniciativas apresentadas em [26] demonstram que estas preocupações já são reconhecidas também pelos usuários finais. Isto ocorre no momento em que os ataques a usuários finais tornam-se mais corriqueiros e notadamente mais complexos e difíceis de detectar.

A disponibilização de ferramentas de segurança destinadas aos usuários finais tem se tornado mais comum. A criação de *suites* de segurança voltados a este tipo de usuário, assim como a disponibilização de ferramentas (*firewalls*, a*ntispywares* e antivírus) acoplados ao sistema operacional, também tem auxiliado na segurança do usuário final. No entanto, estas ferramentas têm como objetivo evitar o ataque, mas não identificar a ocorrência de um ataque e tomar contramedidas contra o invasor.

A principal contribuição deste trabalho é disponibilizar ao usuário final uma ferramenta de segurança, usualmente, aplicada em grandes organizações: o IDS (Sistema de Detecção de Intrusos). Propomos essa ferramenta através da criação do modelo de IDS-NIDIA Remoto. O modelo de IDS-NIDIA Remoto, tem como objetivo fundamental utilizar a capacidade de detecção de um IDS existente, através da extensão e adaptação do mesmo, para proteger usuários finais que não possuem este tipo de ferramenta a disposição.

O modelo de IDS-NIDIA Remoto foi apresentado através da explanação dos componentes que o integram. Outra contribuição foi a utilização de um conjunto de tecnologias não usualmente integradas. Através do desenvolvimento do Modelo de IDS-NIDIA Remoto e da sua implementação, fez-se a integração de *Web services*, Sistemas

Multiagentes e MDA (através do repositório de metadados), obtendo-se desta forma, conhecimentos importantes para utilização destas tecnologias.

Um protótipo do modelo do IDS-NIDIA Remoto foi desenvolvido. Isto foi feito através da adaptação e extensão do IDS-NIDIA e através da implementação de protótipos dos componentes do modelo (IDS-Client, IDS-Proxy, NIDIA Profiles). Desta forma, apresentamos de forma prática a utilização e integração das tecnologias inerentes ao modelo, assim como, a aplicação do modelo no seu objetivo de proteger um usuário final sob um ataque de invasão.

Desta forma, apresentamos um IDS que é:

- Provido como um *Web service*, então é facilmente usado por usuários externos conectados na Internet;
- É baseado em agentes inteligentes que usam uma rede neural para análise, então podem aprender e usar o conhecimento adquirido para tomar contramedidas mais eficientes;
- É baseado em *Web services* e MDA, então pode ser facilmente integrado com outras ferramentas.

6.2 Considerações Finais

Este trabalho buscou enfatizar a necessidade de um IDS Remoto e importância do uso das tecnologias *Web services*, MDA e Sistemas Multiagentes no desenvolvimento de ferramentas de segurança destinadas a proteger o ambiente computacional do usuário final. Durante o desenvolvimento desta pesquisa, constatou-se que a área de pesquisa para o desenvolvimento de soluções de segurança voltadas para os usuários finais é pouco explorada. Não identificamos nas pesquisas de IDS, apresentadas no Capítulo 2 e avaliadas durante o período de escrita, que estes tenham interesse em proteger os usuários além daqueles pertencentes a sua rede local.

A pequena quantidade de pesquisas destinadas ao nosso foco principal (proteger usuários finais) dificultou que fosse feito um plano comparativo entre o nosso modelo proposto e outras ferramentas IDS existentes.

Apesar de todos os recursos informacionais que surgiram nos últimos anos, ainda não existe de forma efetiva e difundida a elaboração de planos de segurança voltados a

proteger exclusivamente os usuários finais. No entanto, em nossa pesquisa constatamos que está sendo cada vez maior o número de usuários finais que se preocupam em ter em seu computador ferramentas como: *firewall*, *anti-spywares* e antivírus.

Constatamos também que os resultados experimentais com as tecnologias envolvidas na pesquisa, associado ao estudo bibliográfico apresentado, contribuem para avaliar a aplicação dessas tecnologias em outros projetos e na continuação da utilização deste.

O projeto NIDIA, utilizado por nossa pesquisa como ferramenta IDS no Modelo de IDS-NIDIA Remoto, tem recebido novas atualizações decorrente das novas pesquisas realizadas. As pesquisas têm como finalidade propor soluções para a comunicação segura e confiável entre os agentes do sistema [50], a utilização de mensagens baseadas em XML entre os agentes e a utilização das bases de dados em XML [12].

6.3 Limitações

Apesar de alcançarmos pontos positivos com nosso trabalho alguns pontos não poderam ser ultrapassados, indicando algumas limitações em nosso modelo proposto.

Algumas limitações que podemos citar são:

- No processo de captura de informações do sistema do usuário final, o IDS-NIDIA Remoto apresentou problemas de execução em seu módulo cliente (IDS-Client) quando utilizado em conjunto com ferramentas proprietárias como antivírus, firewalls e anti-spywares. Devido ao módulo de captura de informações do IDS-Client funcionar como *sniffer* de rede, as ferramentas proprietárias identificavam o IDS-Client como uma aplicação prejudicial ao sistema;
- As bases de dados que o IDS-NIDIA Remoto faz uso estão pouco "populadas". É necessário que estas bases sejam atualizadas com uma maior quantidade de informações sobre ataques para que o IDS possa ter uma melhor capacidade de detecção;
- O módulo de contramedidas do IDS-Client foi preparado para um pequeno conjunto de ataques, sendo necessário o acréscimo de novas contramedidas;

- A forma de atualização dos componentes do IDS-Client não é transparente ao usuário final. No IDS-Client, não há nenhum componente responsável por verificar novas atualizações das *facilitys*.

Tais limitações podem ser futuramente sanadas, algumas destas são propostas como trabalho futuro, levando desta forma a um aprimoramento do IDS-NIDIA Remoto.

6.4 Trabalhos Futuros

Os experimentos com as tecnologias de *Web services*, MDA e Sistemas Multiagentes permitiram a construção de um Modelo de IDS Remoto que contempla importantes requisitos para um sistema de detecção de intrusão que pode ser usado por usuários finais.

A aplicação do modelo permitiu definir a metodologia proposta neste trabalho como aplicável para a detecção de intrusões em diferentes ambientes computacionais. As tecnologias empregadas no IDS-NIDIA Remoto permitem ao usuário final fazer utilização deste, para as atividades de detecção, análise, armazenamento e geração de respostas.

A partir dos resultados e contribuições alcançados, outros estudos podem ser realizados, e novas técnicas e métodos podem ser incorporados. Como propostas para trabalhos futuros, nós podemos citar:

- Aplicação desse modelo em outros ambientes computacionais tais como usuários finais usando redes sem fio;
- Estudo e implementação de padrões de reatividade específicos para os usuários finais, tornando-o mais próximo da diversidade provida por este tipo de usuário;
- Permitir que o IDS-NIDIA Remoto trabalhe em conjunto com outras Ferramentas tais como, firewalls, anti-spywares e antivírus. Criando uma *suíte* de ferramentas Remotas destinadas ao usuário final;
- Aplicação de outras técnicas inteligentes para a detecção e análise de eventos, tais como mineração de dados, devido a heterogeneidades dos dados obtidos pela variedade dos sistemas dos usuários finais.
- Estudo e implementação de técnicas de escalabilidade aplicadas aos componentes do sistema;

- Estudo e aplicação de técnicas voltadas à análise da performance do sistema, aplicando o protótipo em um ambiente de rede “stressado”;
- Introduzir a comunicação segura e confiável entre o IDS-Client e o IDS-Proxy.

Essas propostas visam aumentar a eficácia do modelo de detecção de intrusão para torná-lo mais dinâmico quanto às tarefas de reconhecimento de intrusões e elaboração de planos especializados de respostas.

Referências Bibliográficas

[1] ALMEIDA, H. O.; PERKUSICH, A.;COSTA, E. B; PAES, R. B. **COMPOR: a Methodology, a Component Model, a Component based Framework and Tools to Build Multiagent Systems.** CLEI Electronic Journal, 7(1), 2004.

[2] AGENT TECHNOLOGY GROUP NORTEL NETWORKS. **FIPA-OS Feature Overview.** Disponível em: < http://fipa-os.sourceforge.net/docs/presentations/FIPA-OS Feature Overview.pdf >. Acesso em: 8 set. 2006.

[3] AGENT ORIENTED SOFTWARE GROUP. **JACK Documentation.** Disponível em < http://www.agent-software.com/shared/resources/index.html >. Acesso em: 8 set. 2006.

[4] ANDERSON, J.P. **Computer Security Threat Monitoring and Surveillance**. James P. Anderson Co., FortWashington, PA, April 1980.

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

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

[7] BAUER, D. S.; KOBLENTZ, M. E. **NIDX - An Expert System for Real-Time Network Intrusion Detection**, In : Proceedings of IEEE Computer Network Security Symposium, 1988. p. 98-106.

[8] BRADSHAW, J. M. **An introduction to software agents**. In: BRADSHAW, J. M. (Ed.). Software Agents. Massachusetts: MIT Press 1997.

[9] BRENNER, W.; RÜDIGERr, Z.; WITTIG, H. **Intelligent Software Agents. Foundations and applications**. Berlin: Springer-Verlag, 1998.

[10] CHAVES, Marcelo H. P. C. **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.

[11] CHRISTENSEN, Erik; CURBERA, Francisco; MEREDITH, Greg; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001.** Disponível em: < http://www.w3.org/TR/wsdl >. Acesso em: 8 set. 2006.

[12] CLAUDINO, Emanoel; ABDELOUAHAB, Z. (2006). **Management and Integration of Information in Intrusion Detection System: Data Integration System for IDS Based Multi-Agent Systems.** International Workshop on Interaction between Agents and Data Mining (IADM-06). 2006.

[13] COLLIS, Jaron; NDUMU, Divine; BUSKIRK, Christopher van. **The Zeus Technical Manual. Release 1.04, July 2000.** Disponivel em: < http://labs.bt.com/projects/agents/zeus/techmanual/TOC.html >. Acesso em: 8 set. 2006.

[14] CUTLER, Roger; DENNING, Paul. **List of Web Services Specifications Compiled by Roger Cutler and Paul Denning.** Disponível em: <http://lists.w3.org/Archives/Public/www-ws-arch/2004Feb/0022.html>. Acesso em: 8 set. 2006.

[15] DARPA - KNOWLEDGE SHARING INITIATIVE EXTERNAL INTERFACES WORKING GROUP. **Specification of the KQML Agent-Communication Language plus example agent policies and architectures.** Disponível em <http://www.cs.umbc.edu/kqml/kqmlspec/spec.html >. Acesso em: 8 set. 2006.

[16] DENNING, D. **An Intrusion Detection Model.** IEEE Transactions on Software Engineering 13, 2 (Feb.), 1987. p. 222-232.

[17] 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.

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

[19] FININ, Tim; LABROU, Yannis; MAYFELD, James. **KQML as an agent communication language.** IN: BRADSHAW, Jeffrey (Ed.). Software Agents. Menlo Park: AAAI Press/The MIT Press, 1997. p.291-316.

[20] FIPA. **Foundation for Intelligent Physical Agents.**, Disponível em: < http://www.fipa.org/>. Acesso em: 8 set. 2006.

[21] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. **FIPA ACL Message Structure Specification.** Disponível em < http://www.fipa.org/specs/fipa00061/index.html >. Acesso em: 8 set. 2006.

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

[23] GLUZ, J. C.; VICCARI, R. M. **Linguagens de Comunicação entre Agentes: Fundamentos Padrões e Perspectivas.** In: III Jornada de Mini-Cursos de Inteligência Artificial - SBC2003, 2003. p. 53-102.

[24] GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean-Jacques; NIELSEN, Henrik Frystyk. **SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation 24 June 2003**. Disponível em: <http://www.w3.org/TR/soap12-part1/>. Acesso em: 8 set. 2006.

[25] HEGAZY, Islam; ALARIF, Taha; FAYED, Zaki T.; FAHIM, Hossam M. **A framework for multiagent-based system for intrusion detection.** In the proceedings of the 3rd International Conference in Intelligent Systems Design and Applications, Agosto 2003.

[26] HOEPERS, Cristine; STEDING-JESSEN, Klaus; CORDEIRO, Luiz E. R; CHAVES, Marcelo H. P. C. **A National Early Warning Capability Based on a Network of Distributed Honeypots**, 17th Annual FIRST Conference on Computer Security Incident Handling, Singapore, Junho, 2005.

[27] HTTP WORKING GROUP INTERNET DRAFTS. **HTTP Specifications and Drafts**. Disponível em:< http://www.w3.org/Protocols/Specs.html >. Acesso em: 8 set. 2006.

[28] IETF - INTERNET ENGINEERING TASK FORCE. **SSL 3.0 Specification.** Disponível em: <http://www.freesoft.org/CIE/Topics/ssl-draft/3-SPEC.HTM>. Acesso em: 8 set. 2006.

[29] IKV. **Grasshopper - A Plataform for Mobile Software Agents.** IKV, 1999. Disponível em: <http://www.ikv.de/products/grasshopper>. Acesso em: 8 set. 2006.

[30] JCP. **Java Metadata Interface API 1.0 Specification, 2004**. Disponível em: <http://java.sun.com/products/jmi>. Acesso em: 8 set. 2006.

[31] JPCAP. **JPCAP (Java Package for Packet Capture), 2006.** Disponível em: <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/document.html>. Acesso em: 8 set. 2006.

[32] KRUGEL, Christopher; TOTH, Thomas. **Applying mobile agent technology to intrusion detection.** In ICSE Workshop on Software Engineering and Mobility, 2001.

[33] KRUGEL, Christopher; TOTH, Thomas. **Sparta - a security policy reinforcement tool for large networks.** In submitted to I-NetSec 01, 2001.

[34] KURTZ, George; SCAMBRAY, Joel; MCCLURE, Stuart. **Hackers Expostos - Segredos e Soluções.** 4. ed. São Paulo: Campus Brasil, 2003. 832 p

[35] LABROU, Y.; FININ Tim; PENG, Yun. **Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems**. 1999. p. 45-52.

[36] 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.

[37] MATULA, M. **NetBeans Metadata Repository - WhitePaper, 2003**. Disponível em: <http://mdr.netbeans.org/docs.html>. Acesso em: 08 de agosto de 2006.

[38] MIT TEAM MEMBERS. **Kerberos: The Network Authentication Protocol.** Disponível em <http://web.mit.edu/Kerberos/>. Acesso em: 8 set. 2006.

[39] NAKAMURA, Emilio Tissato; GEUS, Paulo Lício de. **Segurança de Redes em ambientes cooperativos**. 1. ed. São Paulo: Berkeley Brasil, 2002. 320 p.

[40] NETWORK AGENTS RESEARCH GROUP. **April Agent Plataform.** Disponível em: < http://sourceforge.net/projects/networkagent >. Acesso em: 8 set. 2006.

[41] NWANA, H. S. **Software Agents: An Overview.Knowledge Engineering Review**. Vol. 11, No 3, pp.1-40,setembro 1996.

[42] OASIS-UDDI. **OASIS UDDI Advancing Web Services Discovery Standard**. Disponível em: < http://www.uddi.org/ >. Acesso em: 8 set. 2006.

[43] OASIS. **OASIS Web Services Security (WSS) TC**. Disponível em: <http://www.oasis-open.org/committees/wss/>. Acesso em: 8 set. 2006.

[44] OASIS. **OASIS Web Services Reliable Messaging (WSRM) Technical Committee.** Disponível em: <http://www.oasis-open.org/committees/wsrm/>. Acesso em: 8 set. 2006.

[45] OMG. **Model Driven Architecture (MDA)**, OMG document ormsc/01-07-01, 2001. Disponível em: <http://www.omg.org/mda>. Acesso em: 8 set. 2006.

[46] OMG. **Unified Modeling Language (UML) specification**. Disponível em:<http://www.omg.org/technology/documents/formal/uml.htm>. Acesso em: 8 set. 2006.

[47] OMG. **Meta-Object Facility (MOF) specification, OMG document ad/03-02-08.** Disponível em: <http://www.omg.org/technology/documents/formal/mof.htm>. Acesso em: 8 set. 2006.

[48] OMG. **Common Warehouse Metamodel (CWM) specification.** Disponível em: <http://www.omg.org/technology/documents/formal/cwm.htm>. Acesso em: 8 set. 2006.

[49] OMG. **XML Metadata Interchange (XMI) specification** Disponível em: <http://www.omg.org/technology/documents/formal/xmi.htm>. Acesso em: 8 set. 2006.

[50] OLIVEIRA, Emerson. J. S.; LOPES, Denivaldo; ABDELOUAHAB, Zair. (2006). Security on MASs with XML Security Specifications. 17th International Conference on Database and Expert Systems Applications (DEXA'06), 2006. p. 5-9.

[51] 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.

[52] POSTEL, Jonathan B. **SMTP - SIMPLE MAIL TRANSFER PROTOCOL - RFC 821.** Disponível em: < http://www.ietf.org/rfc/rfc0821.txt >. Acesso em: 8 set. 2006.

[53] ROESCH, M. **Snort: Lightweight intrusion detection for networks.** In Proceedings of the 13th Conference on Systems Administration, Seattle, USA, November 1999.

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

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

[56] SILVA, Mauro;LOPES, Denivaldo; ABDELOUAHAB,Zair. **A Remote IDS based on Multi-agent Systems, Web Services and MDA**. International Conference on Software Engineering Advances. ICSEA 2006.

[57] SLAGELL, M. **The Design and Implementation of MAIDS (Mobile Agents for Intrusion Detection System)**, Masters Creative Component paper, Iowa State University, May 2001.

[58] SMAHA S. **Haystack: An Intrusion Detection System.** Proceedings of the IEEE forth Aerospace Computer Security Applications Conference, Orlando, Florida, 1988.

[59] STOLFO, S.J.; PRODROMIDIS, A.L.; TSELEPIS, S.; LEE, W.; FAN, D.; 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, 1997, pp. 74–81.

[60] 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.

[61] SUN MICROSYSTEMS, INC. **The Java Web Services Tutorial 1.3.** Disponível em <http://java.sun.com/webservices/downloads/webservicestutorial.html>. Acesso em: 8 set. 2006.

[62] SYSTINET CORPORATION. **Systinet White Papers.** Disponível em: <http://www.systinet.com/resources/white_papers>. Acesso em: 8 set. 2006.

[63] SYMANTEC ENTERPRISE SECURITY. **Symantec Internet Security Threat Report Trends for January 06–June 06 Volume X**, Setembro, 2006.

[64] SERVILLA, Mark; HEADY, Richard; LUGER, George; MACCABE, Arthur. **The architecture of a network level intrusion detection system.** Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990.

[65] TILAB. **JADE - Java Agent Development Framework.** Disponível em: < http://jade.tilab.com/ >. Acesso em: 8 set. 2006.

[66] TILAB. **JADE Socket Proxy Agent.** Disponível em: <http://jade.tilab.com/doc/tools/SocketProxyAgent/index.html>. Acesso em: 8 set. 2006.

[67] WEISS, Gerhard; **Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence**, The MIT Press - Cambridge, Massachusetts, London, England, 1999.

[68] WEB SERVICES ARCHITECTURE WORKING GROUP. **Web Services Activity.** Disponível em: <http://www.w3.org/2002/ws/>. Acesso em: 8 set. 2006.

[69] WEB SERVICES ARCHITECTURE WORKING GROUP. **Web Services Architecture W3C Working Group Note 11 February 2004**. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em: 8 set. 2006.

[70] W3C. **Web Services Policy 1.2 - Framework (WS-Policy).** Disponível em: <http://www.w3.org/Submission/WS-Policy/>. Acesso em: 8 set. 2006.

[71] W3C. **Web Services Addressing (WS-Addressing) W3C Member Submission 10 August 2004.** Disponível em: <http://www.w3.org/Submission/ws-addressing/>. Acesso em: 8 set. 2006.

[72] WEB SERVICES BUSINESS ACTIVITY FRAMEWORK (WS-BusinessActivity), 2001-2004 BEA Systems Inc, IBM Corporation, Microsoft Corporation.

[73] WS-I. **Web Services Interoperability Organizations.** Disponível em: <http://www.ws-i.org/>. Acesso em: 8 set. 2006.

[74] XML COORDINATION GROUP. **Extensible Markup Language (XML)**. Disponível em: <http://www.w3.org/XML/>. Acesso em: 8 set. 2006.

[75] XU H.; SHATZ S.M. **ADK: An Agent Development Kit Based on a Formal Design Model for Multi-Agent Systems.** Automated Software Engineering, Volume 10, Number 4, 2003. p. 337-365.

[76] ZAMBONELLI, F; PARUNAK,V. **Towards a Paradigm Change in Computer Science and Software Engineering: a Synthesis.** The Knowledge Engineering Review, 18(4): 2004. p. 329-342.