Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências da Computação e Estatística

2° Relatório de Iniciação Científica

"Especificação e Implementação de Agentes Móveis em um Sistema de Detecção de Intrusão"

Processo 99/12226-4

Francisco Gomes Milagres
(francisco@milagres.com**)**

Orientador:

Prof. Dr. Edson dos Santos Moreira
(edson@icmc.sc.usp.br**)**

USP - São Carlos
Junho de 2001

Sumário

1 Resumo

O projeto "*Especificação e Implementação de Agentes Móveis em um Sistema de Detecção de Intrusão*" apresenta a modelagem, implementação e validação de um dos cenários de execução necessários para a constituição de um Sistema de Detecção de Intrusões (SDI) baseado em Agentes Móveis: *Identificação de Usuários Anômalos*, de acordo com a especificação no projeto de pesquisa (Bernardes, 1999).

Este relatório descreve a execução dos segmentos finais do cronograma final de trabalho deste projeto de Iniciação Científica, como mostra o diagrama abaixo:

	Agosto/2000	**Setembro/2000**	**Outubro/2000**	**Novembro/2000**	**Dezembro/2000**	**Janeiro/2001**	**Fevereiro/2001**	**Março/2001**	**Abril/2001**	**Maio/2001**	**Junho/2001**
1. Análise de Requisitos √	■	■									
2. Modelagem do Cenário Proposto √			■	■							
3. Implementação do Cenário Proposto √					■	■	■				
4. Avaliação do Cenário Implementado								■			
5. Validação do Cenário Implementado									■	■	
6. Relatório Final											■

Devido à necessidade de antecipação do relatório final para solicitação de renovação da bolsa de pesquisa, este cronograma foi cumprido com dois meses de antecedência. No capítulo 8 serão explanados os objetivos que justificam a solicitação da renovação da bolsa e um cronograma para o novo período de trabalho será fixado.

2 Introdução

Resultados das pesquisas mais recentes que mostram o cenário da segurança da informação em empresas, instituições de pesquisa e universidades (CSI, 2001) (Módulo, 2000) destacam a cada ano a crescente preocupação com intrusões de origem interna e a mobilidade dos atacantes.

Seguindo a tendência da necessidade de mobilidade também na defesa e reação contra os ataques que cada vez estão evoluindo, surgiu a integração de soluções de segurança como firewalls e sistemas de detecção de intrusões com agentes móveis de software.

Este projeto tem como objetivo a modelagem de um cenário para Identificação de Usuários Anômalos no Sistema de Detecção de Intrusões baseado em agentes móveis modelado por Mauro Bernardes (Bernardes, 1999).

Este é o segundo relatório do projeto "*Especificação e Implementação de Agentes Móveis em um Sistema de Detecção de Intrusão*" está organizado a partir deste ponto, da seguinte forma:

O capítulo 3 apresentará as conclusões da primeira etapa desta pesquisa com um resumo da especificação dos requisitos e a modelagem para o cenário de *Identificação de Usuários Anômalos* no *Sistema de Detecção de Intrusões* em questão.

No capítulo 4 será feita a avaliação dos servidores de agentes e a justificativa técnica para a escolha do ASDK 1.1 como a versão utilizada para este projeto, usando como base a referência do pesquisador Pereira Filho (Pereira Filho, 2001), que também colaborou com este trabalho. A inserção deste capítulo responde às considerações feitas no parecer do avaliador acerca da escolha do ASDK neste projeto.

O capítulo 5 apresentará a implementação e os testes do projeto em Java (JDK, 2001) em conjunto com o ASDK (*Aglets Software Development Kit*) (ASDK, 2001). Serão exibidos trechos de códigos explanados e cópias de telas de execução para a validação do sistema implementado.

O capítulo 6 destaca as atividades realizadas extra-plano durante o desenvolvimento deste trabalho. O 7º capítulo contém as considerações finais e conclusões desta pesquisa.

A solicitação de renovação da bolsa de pesquisa e o cronograma adicional de projeto estão no capítulo 8. O 9º capítulo lista as referências utilizadas nesta fase da pesquisa.

3 Resumo das etapas anteriores

Na etapa inicial deste projeto, foi realizada a Análise dos Requisitos, a Modelagem e o início da Implementação do Cenário de *Identificação de Usuários Anômalos* no *Sistema de Detecção de Intrusões* em questão. A seguir será feito um breve resumo destas etapas para introduzir a segunda parte do trabalho, que é o conteúdo deste relatório.

A primeira das fases do projeto foi a Análise dos Requisitos, proposta em conjunto com trabalho desenvolvido em pesquisa que este trabalho se baseia, de Mauro César Bernardes (Bernardes, 1999). O passo inicial foi identificar os casos de uso do ambiente de *Identificação de Usuários Anômalos* do Sistema de Detecção de Intrusões (SDI) para então, partindo deste ponto, ter uma visão mais precisa das funcionalidades que o sistema deveria realmente suportar para que decisões mais explícitas possam ser tomadas durante as fases de projeto e implementação.

Em seguida, foi Modelado o cenário de Identificação de Usuários Anômalos para o Sistema de Detecção de Intrusões baseado em agentes móveis. A modelagem prevê o uso de pequenos agentes móveis para desempenharem todas as ações de monitoria, tomada de decisão, notificação e reação a tentativas de intrusão. Cada agente então opera de forma independente um dos outros, porém, de forma cooperativa, formando um completo sistema de detecção de intrusões.

Através de uma arquitetura em 4 camadas (Vigilância, Tomada de Decisão, Notificação e Reação), cada uma delas representando um grupo de tarefas específicas desempenhadas por agentes especializados e que se comunicam através do mecanismo de troca de mensagens, o cenário de identificação foi definido. Então, com base em informações coletadas pelos *Agentes de Vigilância*, *Agentes de Tomada* de Decisão entrarão em ação, analisando e identificando possíveis intrusões. Caso uma ação seja considerada suspeita por estes agentes, *Agentes de Notificação* serão acionados e cuidarão de notificar o administrador da rede ou acionar os agentes de nível superior. Em último nível, encontram-se os *Agentes de Reação*. Estes agentes cuidarão de contra-atacar automaticamente as possíveis intrusões, com base nas informações dos agentes de notificação ou ainda, serem acionados através de uma intervenção do administrador da rede.

A modelagem do Cenário de Identificação de Usuários Anômalos de forma gráfica, através de um DFD, destaca também a interface feita do SDI com o administrador do sistema e com o sistema externo ao de detecção de intrusões e é ilustrada na figura 1.

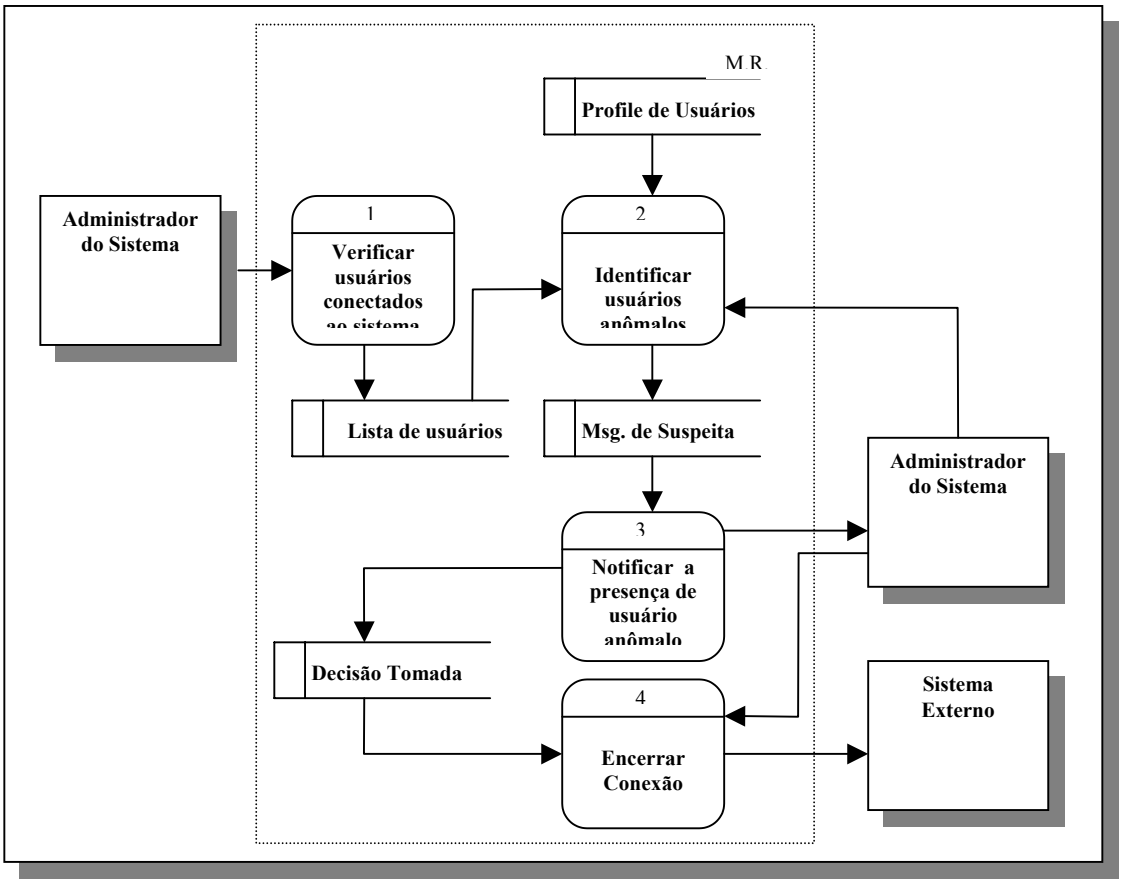

Figura 1 Modelagem do Cenário de Identificação de Usuários Anômalos

A introdução do conceito dos Aglets foi o tema final da primeira etapa do trabalho. Através do uso da linguagem Java e da API Aglet ASDK – que é um kit de desenvolvimento de agentes móveis – , que foi a escolhida para este trabalho.

4 Avaliação do uso de Aglets

De acordo com o relatório da fase inicial deste projeto, foram desenvolvidas no grupo de pesquisa análises de viabilidade de implementação dos agentes móveis usando a plataforma ASDK (*Aglets Software Development Kit*) da IBM (ASDK, 2001), que possibilitou a implementação dos Aglets em linguagem Java.

A estratégia da pesquisa e os resultados para a implementação de agentes móveis em um Sistema de Detecção de Intrusão (SDI) foram extraídos dos resultados da pesquisa de (Pereira Filho, 2001) e (Bernardes, 1999) e a seguir são resumidos os resultados considerados para este relatório.

Estas análises têm a finalidade de demonstrar as características de cada plataforma e seus agentes para que se tenham parâmetros que justifiquem a decisão da indicação da plataforma que mais se adapte ao sistema de detecção de intrusão, neste caso, a ASDK. Os resultados dos testes foram apresentados como trabalho final para obtenção do título de Mestre pelo pesquisador Pereira Filho, que colaborou com esta pesquisa.

Para efetuar os testes necessários na a captura de informações para análise foi necessário compreender o desenvolvimento e a criação de agentes, além de compreender o funcionamento de cada plataforma avaliada.

Dentre as plataformas avaliadas, Concordia (Mitsubishi, 1997) e Gossip (Tryllian, 2000) foram as que mais apresentaram problemas, pois suas documentações ainda estão incompletas, em momento algum foi encontrado um material que explicasse como criar um agente básico como o que foi encontrado nas outras plataformas.

No entanto, utilizando o ambiente Grasshoper (IKV, 2000) e o ASDK (ASDK, 2001), o processo de aprendizagem e desenvolvimento foi bem menos trabalhoso. Todas as plataformas apresentaram uma documentação completa das APIs de desenvolvimento de seus agentes.

4.1 Ambiente de testes

A principal consideração para a execução dos testes e coleta dos resultados foi a criação de um ambiente que não fosse tendencioso a nenhuma das plataformas. Como

há comunicação entre servidores, o ideal seria configurar uma rede onde o tráfego fluísse somente entre as máquinas. Um outro cuidado tomado foi deixar ativos somente os serviços mínimos necessários para evitar concorrência entre outros processos. A disposição e as configurações das máquinas são demonstradas na figura 2. Para testes de desempenho foi utilizada a Máquina 2. A Máquina 1 foi utilizada somente para disparar os agentes para a Máquina 2.

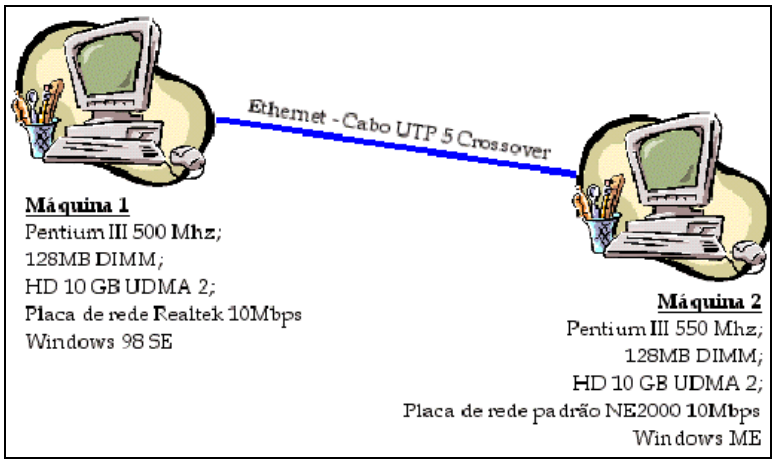

Figura 2 Máquinas em rede para testes

Para a medir os dados de processamento foi utilizado o software Monitor de Sistema, do Windows. Os resultados da monitoração eram armazenados em um arquivo de *log* com as informações de porcentagem de processamento utilizado a cada segundo.

4.2 Análise de desempenho

O objetivo desta análise é demonstrar o desempenho dos agentes de cada plataforma e também comparar os gastos de processamento que os servidores têm quando executam um agente.

A análise de gastos de processamento é importante para que se possa verificar a carga que o servidor sofrerá quando um sistema baseado na plataforma avaliada for utilizado. O interessante é que o sistema faça o menor uso possível do poder de processamento do seu *hos*t, possibilitando o compartilhamento do processador com outras aplicações também relevantes. Para o sistema de detecção de intrusões, o importante é que o usuário não sinta que sua máquina está sendo monitorada e que seu processador está sendo utilizado por aplicativos irrelevantes para o seu trabalho.

Já na análise de desempenho dos agentes, o que se verifica é o tempo de resposta do sistema e o tempo gasto pelo agente para a realização de uma tarefa. Para o SDI, a importância do desempenho dos agentes está diretamente relacionada ao tempo de detecção e reação a uma invasão. A lentidão do processo de reação, captura de informações e de análise pode acarretar na ineficiência do sistema e possivelmente na falta de confiabilidade das informações. Em se tratando de segurança, quanto mais rápido for o resultado da detecção e da reação, mais confiável e eficiente o sistema será. O processo de invasão pode ser demorado, mas, após o fato ser consumado, a ação do atacante pode ser bem mais rápida e quanto mais demorada for a detecção e a reação, menores serão as chances de evitar danos ao sistema invadido.

O processo de simulação de uma comunicação entre servidores utilizou o esquema apresentado anteriormente. A Máquina 1 envia um agente que executa uma multiplicação de matrizes 300 x 300 na Máquina 2. A multiplicação de matrizes foi escolhida, pois possui um código simples e apresenta uma alta demanda de processamento (Figueiredo, 2000).

Para cada plataforma, foi criado um agente que executa o cálculo de multiplicação de matrizes. Para a compilação dos agentes foi utilizado o JDK 1.1.8 para manter a compatibilidade. É importante destacar que somente o ASDK possuía até o momento versão compatível com Java 2, no entanto, a anterior foi a considerada para os testes bem como para a implementação deste trabalho.

Na implementação dos agentes de teste, foi seguido o algoritmo apresentado a seguir:

```
Algoritmo

Declare inicio, fim: inteiro longo
Declare data: Objeto Data
Declare matriz: Objeto Matriz
Declare resultados: literal

// Lendo os resultados já gravados

AbraArquivo("Mult.log");
Enquanto (Não EOF) faça
Resultado LeiaLinhaArquivo("Mult.log");
Fim Enquanto FechaArquivo("Mult.log");

// Tempo do inicio da multiplicação

inicio = data.pegaHora( );

// Multiplicando matrizes

matriz.MULTIPLICA( );
```

```
// Tempo do fim da multiplicação

fim = data.pegaHora( );

// Gravando os resultados concatenando com o resultado atual

AbraArquivo("Mult.log");
EscrevaArquivo(resultados+(fim-inicio));
FechaArquivo("Mult.log");

Fim_Algoritmo.
```

O agente foi então executado 100 vezes para obter um resultado médio mais aproximado do real, diminuindo o risco de influências externas como o próprio *garbage collector* da JVM (Máquina Virtual Java).

4.3 Resultados dos testes

Foram executados os testes nas plataformas ASDK 1.0, ASDK 1.1, Concordia, Grasshopper e Gossip. No momento dos testes, a Tryllian, desenvolvedora do Gossip, disponibilizava um ADK somente para desenvolvedores cadastrados e credenciados, não sendo disponível para *download* gratuito, o que impossibilitou que um agente matriz fosse criado e testado. A inclusão da plataforma nos testes de desempenho teve como objetivo verificar o poder de processamento gasto pelo servidor, pois a interação entre o usuário e os agentes é totalmente gráfica e animada. O objetivo então foi verificar se o fato de possuir uma interface tão amigável não seria prejudicial ao desempenho de sistemas baseados na sua tecnologia.

A metodologia dos testes consistiu na divisão das informações capturadas em 5 etapas:

1) Inicialização do servidor: Processamento gasto para inicializar e ativar o servidor de agentes;

2) Recebimento e instanciação do agente: Processamento gasto para receber o agente e instanciá-lo na memória;

3) Início do agente para execução: Processamento gasto para iniciar a execução do agente (carregamento de todas as classes e reativação do estado pré-transmissão);

4) Execução do cálculo: Processamento gasto para efetuar o cálculo;

5) Encerramento da execução e do servidor: Processamento gasto para encerrar e destruir o agente (*dispose*) e parar a execução do servidor.

As cinco etapas são indicadas nos gráficos de porcentagem de processamento dos servidores para que se tenha uma melhor visualização dos resultados. Foram gerados dois gráficos para cada plataforma avaliada. O primeiro apresenta a porcentagem de utilização do processador e o tempo gasto em cada etapa. O segundo deles apresenta o tempo de cada uma das 100 (cem) execuções do agente matriz.

A primeira versão do ASDK da IBM obteve um bom resultado nos testes, com teste ilustrado a seguir nas figuras 3 e 4.

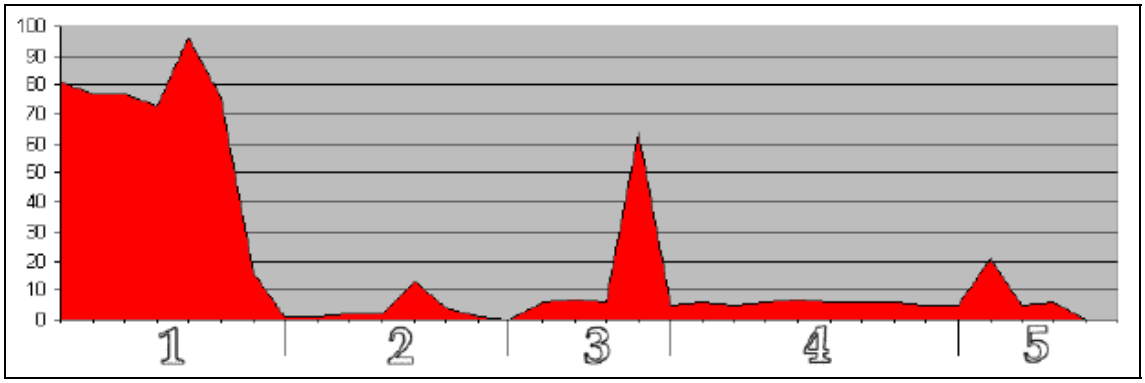

Figura 3 Gráfico de utilização do processador do servidor Tahiti (ASDK 1.0)

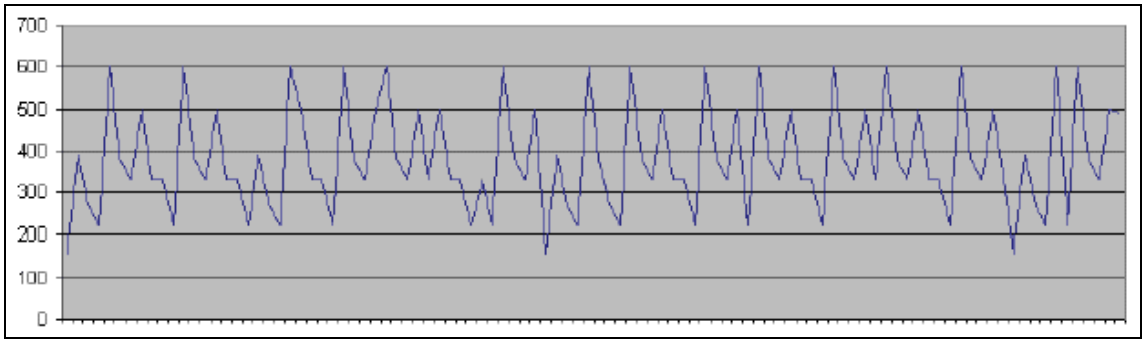

Figura 4 Tempo, em milisegundos, de 100 execuções do agente em ASDK 1.0

A porcentagem média de utilização do processador pelo ASDK 1.0 foi de 21.63%, a segunda mais baixa, perdendo somente para o seu sucessor ASDK 1.1. Já o seu agente se portou de uma maneira instável, alterando entre altos e baixos com a variação de aproximadamente 400 milisegundos e obteve a média de 380.2 milisegundos. Mesmo assim, obteve também a segunda melhor marca quanto ao desempenho do agente, perdendo novamente para o seu sucessor, como poderá ser visto a seguir.

O Tahiti (ASDK 1.1) e seu agente conseguiram os melhores resultados, de acordo com as figuras 5 e 6.

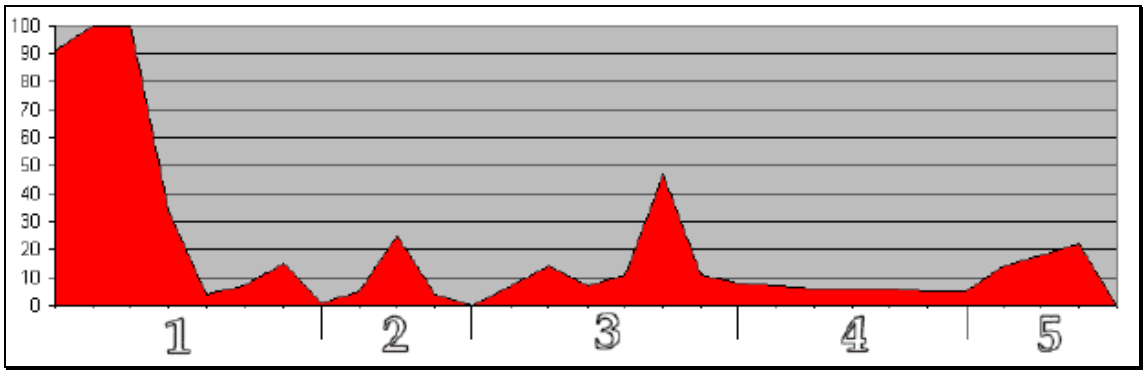

Figura 5 Gráfico de utilização do processador do servidor Tahiti (ASDK 1.1)

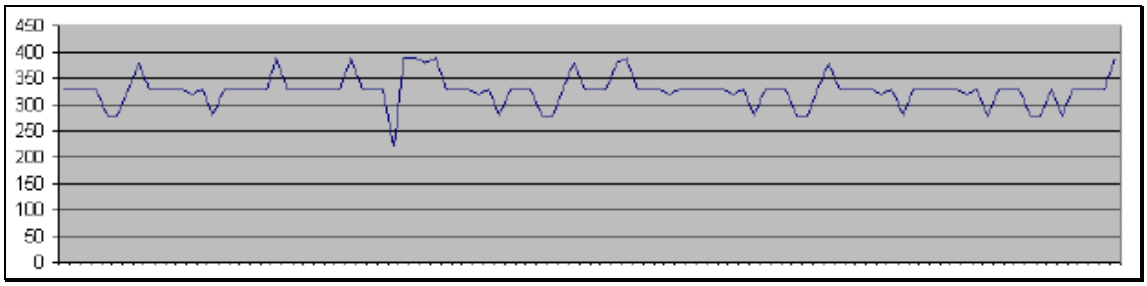

Figura 6 Tempo, em milisegundos, de 100 execuções do agente em ASDK 1.1

A porcentagem média de utilização do processador do servidor neste caso foi de 21%, a menor de todas as plataformas. Já o agente se portou de uma maneira mais estável onde os melhores resultados se diferenciavam do pior em no máximo aproximadamente 150 milisegundos. Sua média de execução foi de 328 milisegundos, se destacando por ser o mais rápido e que exigiu menos poder de processamento.

Resultados com os mesmos níveis de detalhes foram executados também nas outras plataformas de desenvolvimento de agentes, no entanto, não serão aqui totalmente ilustrados por este não se tratarem do escopo principal do presente trabalho e estar contido na íntegra em trabalho usado como referência (Pereira Filho, 2001).

A seguir, tem-se um gráfico geral dos testes de utilização do processador de todas as plataformas (figura 7).

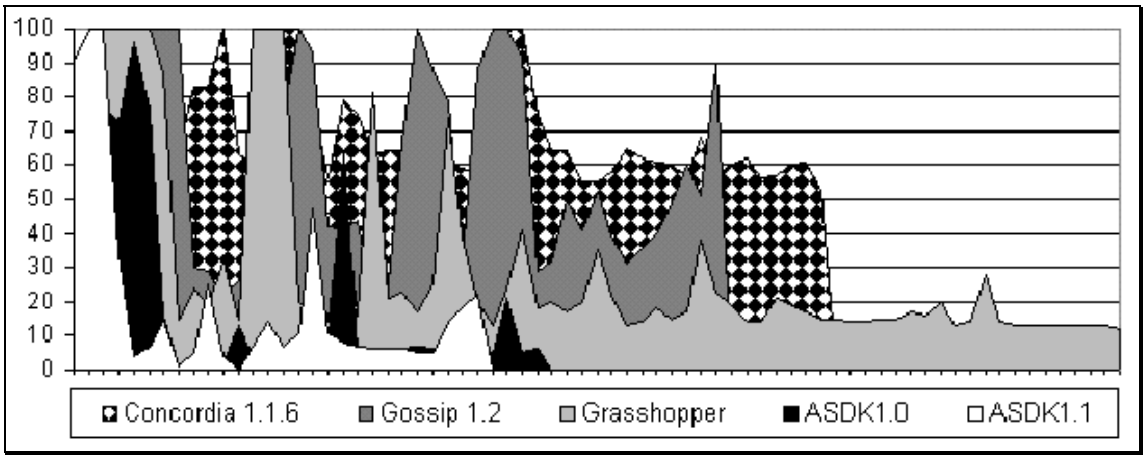

Figura 7 Gráfico geral da taxa de utilização de processamento

O interessante observar é que o Grasshopper consumiu mais tempo para realizar as tarefas enquanto o ASDK 1.1 (Tahiti) gastou bem menos, ou seja, quanto maior é o valor de X (tempo em segundos), maior é o tempo gasto para a realização da tarefa. O outro ponto interessante é a área do gráfico do Concordia, área quadriculada, mostrando que foi o servidor que mais gastou poder de processamento, ou seja, quanto maior o Y (Porcentagem de utilização do processador), mais processamento é utilizado.

O próximo gráfico (figura 8) mostra um plano geral dos resultados de testes de desempenho dos agentes de cada plataforma.

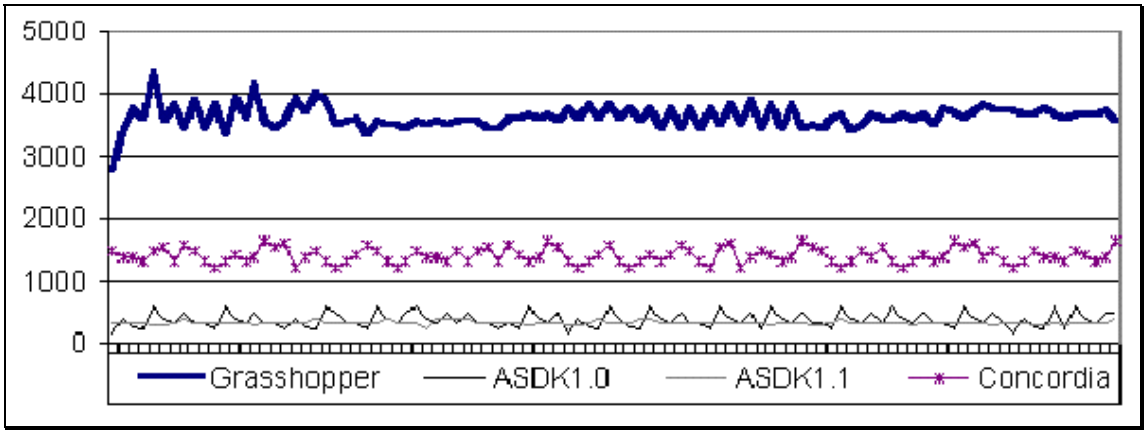

Figura 8 Gráfico geral dos resultados de teste de desempenho dos agentes

O ponto interessante é a linha do gráfico do ASDK 1.1, cinza claro, sobreposta pelo gráfico do ASDK 1.0, pela qual nota-se a uniformidade dos resultados formando quase uma linha reta, mostrando que a previsão do tempo de resposta do sistema tende a ser mais confiável que nos outro sistemas avaliados.

4.4 Análise de Segurança

A segurança em sistemas baseados em agentes móveis envolve quatro problemas distintos:

1. Segurança na transmissão de agentes;

2. Proteção do servidor contra usuários e agentes maliciosos;

3. Proteção do agente contra o ataque de outros agentes;

4. Proteção de agentes contra ataques de servidores hostis.

A utilização da tecnologia de agentes móveis sem levar em consideração aspectos de segurança pode acarretar em grandes catástrofes ao mesmo tempo em que poderia desmotivar sua utilização em tecnologias de risco, como por exemplo, a de detecção de intrusões.

A seguir são apresentados os aspectos de segurança que envolvem as plataformas testadas por Pereira Filho e demonstradas as razões da escolha do ASDK para desenvolvimento deste trabalho. Testes foram executados com a finalidade de detectar falhas de segurança e também apresentar os serviços que são oferecidos para a proteção dos agentes e servidores. Os serviços analisados foram Autenticação de Usuário, de Servidor, de Código e de Agente, Garantia de Integridade, Confidencialida das Informações, Autorização e Auditoria, e que contribuem para a segurança do SDI em desenvolvimento. A figura 9 destaca os ambientes e seus respectivos serviços.

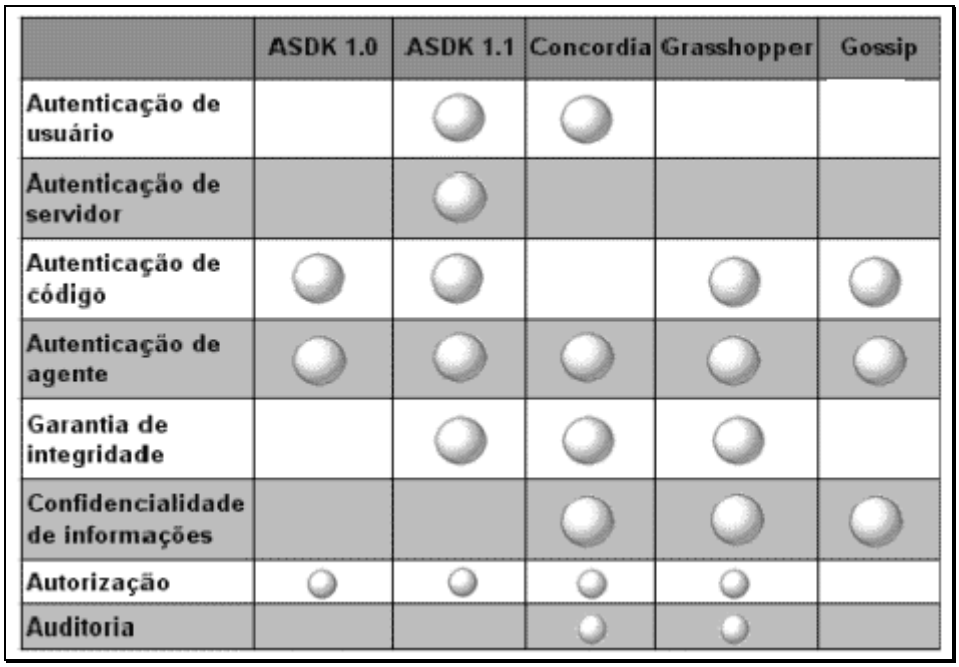

Figura 9 Figura demonstrativa dos serviços de segurança oferecidos pelas plataformas

Com os resultados e conclusões apresentadas aqui, pode-se então apontar quais são os servidores mais seguros para a sua utilização na implementação do sistema de detecção de intrusões baseado em agentes móveis.

A seguir é feito um resumo das características e das conclusões acerca das versões do ASDK 1.0 e 1.1b da IBM e que foram usados neste projeto.

4.5 ASDK 1.0

A versão 1.0 do ASDK da IBM oferece poucos serviços e apresenta diversos problemas de segurança. Dentre os problemas, se destacam:

- Não há autenticação de usuário: um usuário mal intencionado pode criar e enviar agentes maléficos ao sistema podendo apossar-se de informações sigilosas e até mesmo "derrubar" o sistema. Ameaças: Alteração, Acesso ilegal, DoS, Retransmissão.

- Não há autenticação de servidor: são oferecidas bibliotecas para desenvolvimento de servidores, sendo assim, uma pessoa pode criar o seu servidor fazendo-o acessar informações confidenciais dos agentes, altera os dados e enganá-lo. Além disso, o servidor poderá servir de suporte para o usuário atacar outros servidores. Como não há a autenticação a proteção do

sistema pode ser difícil. Ameaças: Alteração, Acesso ilegal, DoS, Retransmissão.

- Não há controle de integridade: o estado ou as informações do agente pode ser alteradas sem que o usuário ou o servidor perceba. Assim, um usuário mal intencionado pode alterar resultados a ponto de lhe favorecer. Ameaças: Alteração, Cavalo de Tróia, Disfarce.

- Não garante a confidencialidade das informações: agentes com informações sigilosas podem ser atacados e seus dados capturados. Dependendo das informações contidas nele, o sistema poderá ser comprometido. O sistema deve proteger o agente tanto na transmissão quanto no seu armazenamento e em nenhuma das duas situações isto é feito. Ameaças: Alteração, Acesso Ilegal.

- Não oferece recursos para auditoria: este serviço é importante para que o administrador possa fazer uma análise do comportamento do sistema para tentar identificar algum problema e até mesmo obter informações necessárias para recuperação do sistema após uma queda ou ataque.

Por outro lado, a versão inicial da plataforma da IBM, oferece alguns serviços de segurança na tentativa de minimizar situações de risco. Os serviços são:

- Autenticação de código: com este serviço, a tentativa é de evitar ataques de agentes que funcionam como cavalo de tróia. Quando o código não é aceito o agente é descartado. O identificador do código é gerado quando o agente é compilado e esta informação é inserida pelo compilador Java.

- Autenticação do agente: quando um agente é compilado com o kit, além do ID que o Java gera para os seus objetos, é também criado um número serial que identifica unicamente o agente. Com isto, a possibilidade que um agente malicioso ataque o sistema é bastante reduzida. Como parte da identificação é considerado o servidor de origem. Os agentes originados do próprio servidor são classificados como confiáveis e os que vêm de fora (de outros servidores) são classificados como não-confiáveis, facilitando a atribuição de permissões e autorização de serviços.

- Serviço de Autorização: mesmo um agente sendo autêntico, ele pode executar tarefas e acessar informações proibidas. Este serviço pode evitar também que agentes não confiáveis abram portas das máquinas para que uma pessoa possa tentar invadir o sistema. A configuração das permissões é feita no servidor Tahiti. As permissões são atribuídas para um grupo de agentes.

4.6 ASDK 1.1 Beta 3

Devido aos diversos problemas de seguranças existentes na versão anterior, a IBM criou o ASDK 1.1 que possui mais serviços, oferecendo menos riscos aos agentes. Mesmo assim, ainda persistem alguns problemas:

- Confidencialidade das Informações: como na versão anterior, o ASDK 1.1 não oferece nenhum serviço que garanta a confidencialidade das informações, tanto nas transmissões quanto no armazenamento dos agentes. Ameaças: Alteração, Acesso ilegal.

- Não oferece recursos para auditoria: não há suporte a serviço de auditoria na plataforma 1.1 do ASDK. Assim, torna-se necessário uma implementação de servidores e agentes que gerem e capture informações para geração de arquivos de *logs* ou algo parecido, minimizando o problema.

Isolando-se estes problemas, a plataforma oferece uma variedade de serviços que auxiliam na criação de sistemas mais seguros e são destacados como vantagens no uso desta versão:

- Autenticação do usuário: o serviço de autenticação de usuário está presente no servidor e é feito sempre quando o servidor é iniciado. Não há um gerenciamento de usuários, sendo somente um para cada servidor, ou seja, não há possibilidade de configurar *profiles* para usuários diferentes.

- Autenticação do servidor: nesta versão pode-se configurar quais servidores podem enviar e receber agentes. A identificação do servidor é feita pelo endereço IP e pelo nome do seu usuário (administrador). As informações são armazenadas em um arquivo e toda vez que há uma transmissão de agentes, elas são verificadas. É uma alteração no protocolo de transmissão de agentes (ATP)

que não existia na versão anterior e foi a única plataforma testada que oferece este serviço.

- Autenticação do código: da mesma forma que o ASDK 1.0, o código é identificado pelo ID que lhe é atribuído quando é utilizado o compilador Java.

- Autenticação do agente: a classificação de confiável e não-confiável segue a mesma regra. A identificação é pela origem do agente e pelo numero serial gerado quando compilado pelo kit. A autenticação de código e agente é importante, pois podem ocorrer casos em que o agente é autêntico, mas pode possuir algum objeto que não é permitido no sistema.

- Garantia de integridade: sempre antes de iniciar a transmissão, quando um *hash* do agente é gerado. Assim, sempre que houver alguma alteração durante a sua transmissão ou armazenamento, esta será detectada. O *hash* é enviado junto com o agente.

- Autorização: seguindo a linha de seu antecessor, o servidor Tahiti também permite atribuir permissões para os grupos de agentes. A diferença é que a possibilidade de atribuir permissões a agentes específicos, aumentando a segurança.

4.7 Escolha da plataforma

Após conclusão do trabalho de pesquisa das características das plataformas de desenvolvimento dos agentes, a escolha foi feita baseada em características desejáveis para os requisitos de segurança e desempenho do sistema de detecção de intrusão em desenvolvimento. Analisando os resultados pode-se observar o seguinte:

- Agentes utilizando ASDK 1.1 são mais rápidos;
- Tahiti do ASDK 1.1 exige menos do processador;

Sistemas implementados utilizando o ASDK 1.1 podem ficar menos vulneráveis aos ataques. Durante os testes, quando foi necessário compreender o funcionamento de cada plataforma, foram observados alguns pontos:

- A plataforma Grasshopper possui uma documentação mais completa;

- O agente Grasshopper é mais fácil para programar e entender, portanto melhor para desenvolvimento;
- O servidor Concordia foi o mais complicado para trabalhar;
- O projeto da IKV++, Grasshopper, está num estágio bem mais avançado do que os concorrentes apresentados. A tarefa de administração e gerenciamento é mais fácil e sua interface é bem intuitiva. O suporte oferecido no site é muito bom.
- A lista de discussão do ASDK 1.0 foi decisiva na implementação do SDI.
- O ASDK 1.1 é o mais seguro e o mais rápido.
- Por ser aberto, o Grasshopper pode ser adaptado para as necessidades tornando-se um servidor mais flexível.

Para o SDI, o importante para a escolha de uma plataforma é verificar se ela apresenta soluções para os problemas de segurança apresentados no item 4.4. Em seguida, as análises de desempenho do servidor e dos agentes influenciam bastante na decisão. Com isto a conclusão tirada é que a plataforma ASDK 1.1 foi a que mais se encaixou nos moldes do sistema. Ela possui a maioria dos serviços necessários para um SDI seguro e não exige muito da máquina na qual está instalada. Além disso, seus agentes apresentaram um desempenho superior aos das outras plataformas avaliadas. Por outro lado, para um sistema mais complexo como Comércio Eletrônico, onde as tarefas de gerenciamento e administração são importantes, o ASDK não é muito útil.

A plataforma que mais se encaixou nestes aspectos foi a Grasshopper. Não só no gerenciamento e administração, como no suporte e na documentação também. A seguir, um tabela comparativa (figura 10) com as características de cada plataforma:

SDI ICMC

Sistemas complexos

	ASDK 1.0	ASDK 1.1	Concordia	Grasshopper	Gossip
Fabricante	IBM	IBM	Mitsubishi	IKV++	Tryllian
Tamanho da plataforma	1.23 MB	1.64 MB	12,6 MB	2,02 MB	2,00 MB
Versão	1.0	1.1 beta 3	1.1 beta 6	2.0	1.2
Java	Java 1	Java 1	Java 1 e 2	Java 1 e 2	Java 1 e 2
Gratuito	Sim	Sim	Não	Sim	Sim
Código Aberto	Não	Sim	Não	Sim	Não
Segurança	★	★★★★½	★★★½	★★★★	★★★
Desempenho	★★★★	★★★★½	★	★★★	★
Tamanho Agente Matriz	2,28 KB	2,28KB	1,66 KB	2,37KB	95,4KB *
Desempenho Agente Matriz	★★★★★	★★★★★	★★★	★★	-
Documentação	★★★½	★★★½	★★	★★★★	★★
Média Geral	★★	★★★★½	★★★	★★★★	★★½

★★★★★ Ótimo ★★★★ Muito Bom ★★★ Bom ★★ Regular ★ Ruim

Figura 10 Tabela comparativa entre todas as plataformas testadas

A tabela destaca a plataformas que foi escolhida para o desenvolvimento do SDI em questão – ASDK 1.1 - e suas características e Grasshopper, sugerida para aplicações complexas. Na classificação final, ASDK 1.1 foi considerado o melhor em segurança, desempenho, seguido pelo Grasshopper e Concodia. Em documentação o destaque foi para o Grasshopper, seguido pelas plataformas da IBM. Na média geral, o melhor foi o ASDK 1.1, justificando assim sua escolha.

5 Implementação

Esta seção apresenta os métodos necessários para o desenvolvimento de um agente móvel simples utilizando-se do conceito de *Aglets.* Esta implementação visa demonstrar a estrutura de um agente *Aglet* para uma melhor compreensão do cenário implementado. Em seguida, será apresentada a implementação dos agentes que compõem o cenário de *Identificação de Usuários Anômalos*.

5.1 Entendendo um Aglet simples

Para uma melhor compreensão da implementação dos agentes contidos neste trabalho, é necessário que se entenda a estrutura básica de um aglet.

Um aglet não é instanciando como um objeto Java qualquer. Todo aglet é criado dentro de um contexto de agentes em um servidor de aglets, no nosso caso o próprio Tahiti. O aglet é sujeito às regras e restrições impostas pelo contexto no qual foi criado. Para que o aglet consiga comunicar com contextos, deve-se criar uma interface entre o aglet e o contexto. Para isso, existe um método no aglet chamado de getAgletContext, que retorna uma interface AgletContext. Essa interface é utilizada pelo aglet para capturar as informações sobre o ambiente e também, para enviar mensagens para o ambiente e outros aglets ativos no ambiente.

Quando um *aglet* é criado pelo contexto, são invocados três métodos:

- O método construtor do aglet;
- O método onCreation, que é executado assim que o método construtor finaliza o seu trabalho;
- E por último, é chamado o método *run***,** que inicia a execução do *aglet.*

Além destes três métodos básicos, o *aglet* possui outros que são importantes no seu funcionamento. Alguns deles são:

- Método dispose¸ utilizado para "destruir" o aglet;
- Método onDisposing, chamado quando é executado o método dispose, muito utilizado para liberar algum recurso alocado pelo aglet antes de sua "destruição";

- E o método handleMessage, que é acionado quando o aglet recebe alguma mensagem do contexto ou outro aglet.

A seguir, é apresentado o código fonte de um *aglet* bem simples que demonstra a utilização dos métodos descritos acima. Todos os métodos básicos de um *aglet* são encontrados no pacote **com.ibm.aglet**. A função deste *aglet* é escrever no console do sistema o nome do método que ele está executando.

```
/** Criando um aglet simples **/
package examples.estudo;
import com.ibm.aglet.*; //Bibliotecas básicas do aglet
public class AgenteSimples extends Aglet { // herdando características de Aglets
/* Método construtor */
public AgenteSimples() {
System.out.println("AgenteSimples: Estou no método construtor!");
}
/* Método executado automaticamente após o construtor */
public void onCreation(Object init) {
System.out.println("onCreation: Acabei de ser criado!");
}
/* Método executado automaticamente após o dispose */
public void onDisposing() {
            System.out.println("onDisposing: Estou liberando os recursos
alocados!");
}
/* Método inicial de execução de um aglet*/
public void run() {
System.out.println("run: Iniciando a execução");
try {
   /* Envia uma mensagem */
   getProxy().sendMessage( new Message("alomundao"));
}
catch (Exception ex) {
       System.out.println("Erro: "+ex);
}
}
/* Método ativado quando o aglet recebe uma mensagem */
public boolean handleMessage(Message msg) {
if (msg.sameKind("alomundao")) { // Verificando a mensagem recebida
 System.out.println("Alomundao");
} else return false;
return true;
} }
```

No código acima, o método **sendMessage** é utilizado para enviar uma mensagem para um *aglet.* Esse, pode ser outro *aglet* qualquer ou ele mesmo. Nesse caso, o *aglet* está enviando uma mensagem para ele mesmo, pedindo para que seja apresentado a mensagem “Alomundao” no console do sistema. A Figura 11 apresenta como fica o console após a execução desse *aglet.*

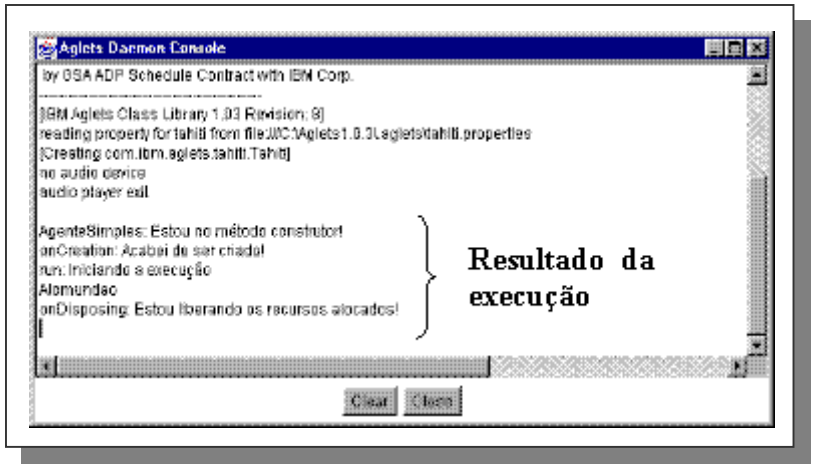

Figura 11 Execução do Aglet simples

5.2 Implementação do Cenário de Identificação de Usuários Anômalos

O cenário escolhido para apresentação neste trabalho, foi o de Identificação de Usuários Anômalos. Este cenário é composto, inicialmente, por cinco agentes dispostos de acordo com a figura 12:

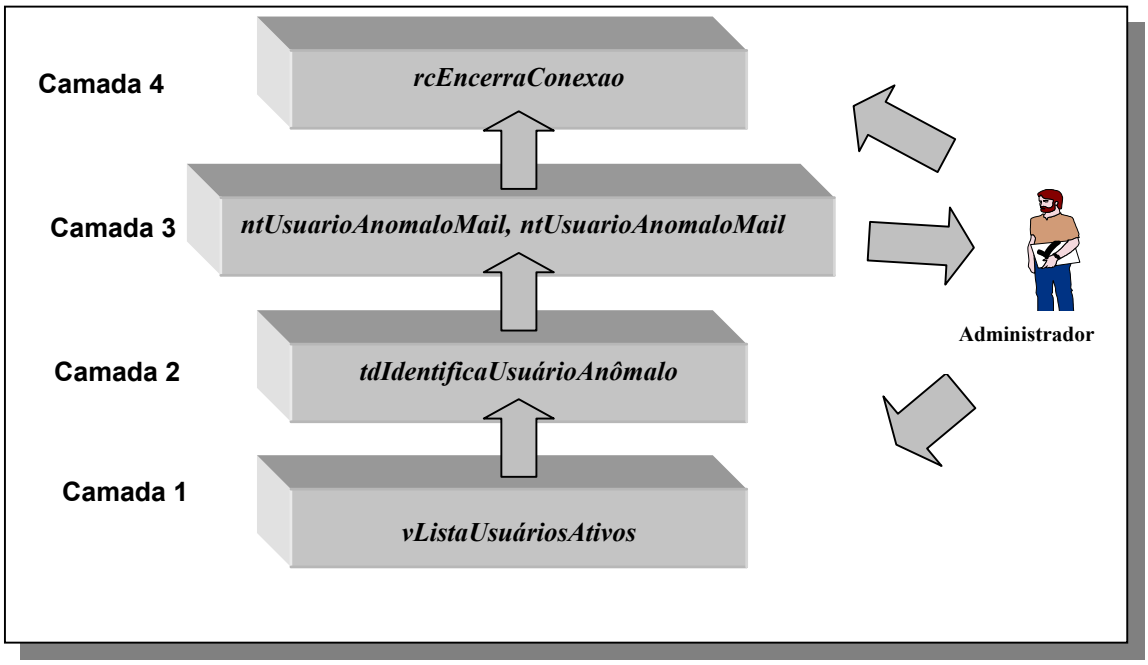

Figura 12 Representação dos Agentes do Cenário de Identificação de Usuários Anômalos

5.3 Agente da Camada de Vigilância

Agente de Vigilância: *vListaUsuáriosAtivos*
Parâmetro: IP da máquina a ser monitorada;
Função: Captura de nomes de usuários "conectados" ao servidor;
Agente ativador: Agente de Controle ou Administrador;
Agente a ser ativado: *tdIdentificaUsuarioAnomalo*.

O agente de vigilância tem a função de trafegar pela rede, nas máquinas que irá monitorar e capturar um lista com informações dos usuários. Exemplificando, se o sistema operacional da máquina de destino for padrão UNIX, o agente executará os comando *w* e *who,* obtendo informações dos usuários que estão "on-line" no sistema. Ao retornar, ele ativa o agente de tomada de decisão enviando estas informações em uma mensagem.

O agente de vigilância pode ser ativado tanto pelo Administrador quanto pelo agente de controle do sistema (figura 13).

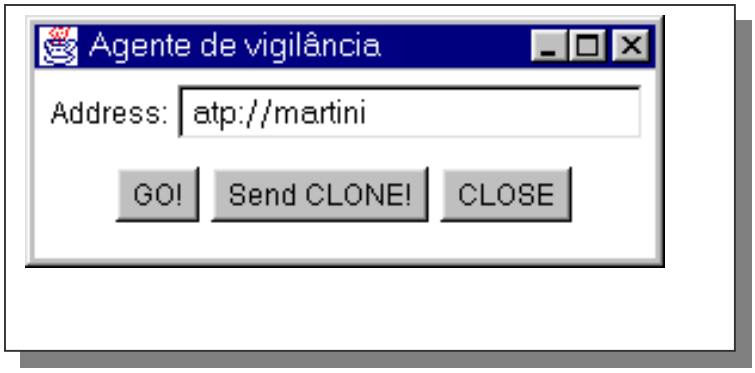

Figura 13 Ativação do agente pelo administrador

A seguir, são apresentados os trechos de código mais importantes deste agente:

```
/* Executa o comando e atribui um processo */
pExecutaComando=Runtime.getRuntime().exec("who");
/* Atribui o stream de resultado ao stream gerado pelo processo */
disResultadoComando = new
DataInputStream(pExecutaComando.getInputStream());
sResultadoProcessado = "";
/* Transfere o que esta no Stream de resultado para uma string */
/* esta string possui o resultado da execução do comando WHO */
while ((sLinhaResultado = disResultadoComando.readLine())!= null) {
             sResultadoProcessado += sLinhaResultado + "\n\r";
}
```

Observamos no trecho acima a instrução `itinerary.go(destination, "execWho")`. O método `go` da classe `itinerary` é responsável por enviar (dispatch) o *Aglet* para o servidor a ser monitorado. O endereço de destino está em `destination` e foi recebido da tela de ativação do agente ou do agente de controle. Esse método envia também a mensagem "`execWho`". Essa irá acionar o método de execução do comando `Who`. Esse irá capturar informações das conexões ativas.

```
SimpleItinerary itinerary = null;

/**** trecho responsável por enviar o agente ao
servidor destino
        try {
            itinerary.go(destination, "execWho");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
/******
         try {
          itinerary.go(home, "atHome");
        } catch (Exception ex) {
            ex.printStackTrace();
}
```

O trecho `itinerary.go(home, "atHome");` contém as informações para, após a execução do agente no servidor remoto, envia-lo de volta o servidor de origem. A mensagem `atHome` será responsável pela ativação do agente tomada de Decisão e a desativação (dispose) do agente de vigilância.

```
public boolean action(Event ev, Object obj) {
        if (ev.target == send) {
            aglet.message = msg.getText();
           try {
              /** criação do clone do Aglet para ser enviado ao servidor*/
              AgletProxy aglet2 = (AgletProxy)aglet.clone();
aglet2.sendOnewayMessage(new message("inicia",address.getText()));
           } catch (Exception e) {
              e.printStackTrace();
           }
       } else if (ev.target == go) {
            aglet.message = msg.getText();
             /*envia uma mensagem para disparar o agente p/ servidor remoto*/
            aglet.handleMessage(new Message("inicia", address.getText()));
        } else if (ev.target == close) {
            hide();
        } else {
            return false;
        }
        return true;
    }
}
```

O ilustração acima apresenta o trecho de envio (*dispach*) do agente de vigilância.

5.4 Agente da Camada de Tomada de Decisão

Agente de Tomada de Decisão: *tdIdentificaUsuárioAnômalo*

Parâmetro: Informações coletadas pelo agente de vigilância;

Função: Verificar a existência de usuários anômalos;

Agente ativador: Agente de Vigilância

Agente a ser ativado: Agente de Notificação para envio de E-mail *ntUsuárioAnômaloMail* e agente de notificação via console *ntUsuárioAnomaloConsole*

O agente desta camada (*tdIdentificaUsuarioAnomalo)* é ativado pelo agente da camada de Verificação (*vListaUsuariosAtivos)* recebendo como parâmetros uma lista contendo informações dos usuários conectados ao sistema. Este agente irá, com base em um arquivo de *profile* verificar possíveis usuários anômalos. Ao suspeitar de uma possível anomalia, este agente irá invocar os agentes de notificação da camada superior.

5.5 Agentes da Camada de Notificação

Neste cenário, esta camada é composta inicialmente por dois agentes. Estes agentes, com base nas informações recebidas do agente de tomada de decisão, irão notificar o administrador da rede da suspeita de um usuário anômalo.

5.5.1 Agente de notificação por e-mail

Agente de Notificação: *ntUsuarioAnomaloMail* (Agente Estático)
Parâmetro: e-mail do administrador;
Função: enviar e-mail ao administrador identificando usuário anômalo;
Agente ativador: *tdIdentificaUsuarioAnomalo*
Agente a ser ativado: *rcEncerraConexão*

Este agente é responsável pela notificação via e-mail e acionamento do agente de reação da camada superior. A seguir, é apresentado o trecho principal de código deste agente.

```
try {
   /* Cria um socket para a porta de SendMail do servidor de e-mail */
   Socket sendMail = new Socket("smtp.milagres.com", 25);
   /* Cria um stream para escrever a mensagem no socket */
   PrintStream mensagem = new PrintStream(sendMail.getOutputStream());
   /* Cria um stream para ler as mensagens do socket */
   DataInputStream in = new DataInputStream(sendMail.getInputStream());
   /* Escrever a mensagem */
   mensagem.println("HELO "+hostEnvio);
   System.out.println(in.readLine()); //Lendo a resposta do servidor
   mensagem.println("MAIL FROM: "+emailAdmin);
   System.out.println(in.readLine());
   mensagem.println("RCPT TO: "+emailAdmin);
   System.out.println(in.readLine());
   mensagem.println("DATA");
   System.out.println(in.readLine());
   mensagem.println("Subject: Notificação de usuário anômalo");
   mensagem.println("Atenção: Às "+horaDetect+" foi detectada uma anomalia para o
"+"usuário "+usuario);
   mensagem.println("Este se conectou ao sistema às "+horaConnect+" através do
"+"endereço "+ipUser);
   mensagem.println("Este usuário foi bloqueado às "+horaDetect);
   mensagem.println("Favor verificar arquivos de logs!");
   mensagem.println("Agente de Notificação via e-mail");
   mensagem.println("\n\r");
   mensagem.println(".");
   mensagem.println("\n\r");
   System.out.println(in.readLine());
   mensagem.println("QUIT");
   System.out.println(in.readLine());
   mensagem.flush();
   sendMail.close(); ...}
```

A figura 14 apresenta a notificação de uma suspeita de intrusão com base em anomalias de usuários recebida pelo administrador do sistema via e-mail e a figura 15 apresenta as mensagens de console para o acionamento do agente, que envia o e-mail.

From: Francisco G Milagres <francisco@milagres.com> 14:25
Subject: Notificação de usuário anômalo
To: security@milagres.com

```
Atenção: Às 12:37 foi detectada uma anomalia para o usuário TESTE1
Este se conectou ao sistema às 12:02 através do endereço 200.206.141.25
Este usuário foi bloqueado às 12:37
Favor verificar arquivos de logs!
Agente de notificação via Email
```

Figura 14 Notificação de uma anomalia recebida via e-mail

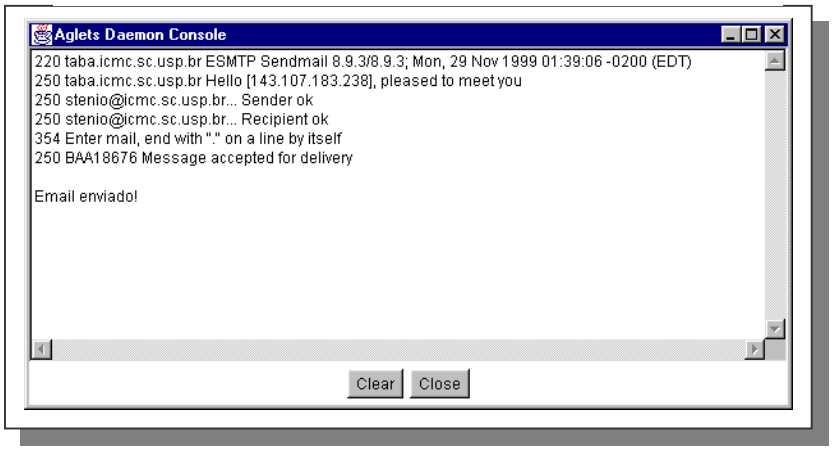

Figura 15 Notificação de uma anomalia sendo enviada por e-mail

5.5.2 Agente de notificação via Console

Agente de Notificação: *ntUsuarioAnomaloConsole*
Parâmetro: IP da máquina do administrador;
Função: notificar o administrador de usuários anômalos via console
Agente ativador: *tdIdentificaUsuarioAnomalo*;
Agente a ser ativado: *Nenhum*

Este agente é responsável pela notificação do administrador do sistema enviando uma mensagem para o console da máquina ao qual ele conectado e é ilustrado através da figura 16.

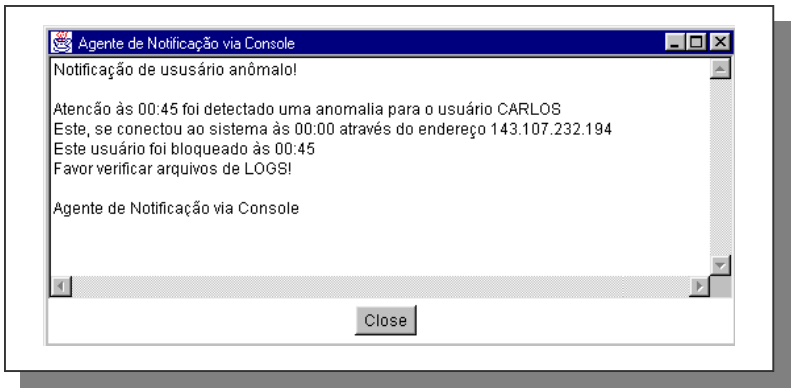

Figura 16 Notificação via console de uma suspeita de anomalia

5.6 Agente da Camada de Reação

Neste cenário, este agente é responsável por encerrar e bloquear a conexão para um usuário identificado como anômalo.

Seu código é muito semelhante ao código do agente de vigilância deste cenário e compreende a execução de um comando *kill* seguido de um bloqueio da senha do usuário.

Agente de Reação: *rcEncerraConexão*
Parâmetro: Informação do usuário anômalo
Função: Derrubar conexão e bloquear acesso deste usuário
Agente ativador: *ntUsuarioAnomaloMail*
Agente a ser ativado: *nenhum*

6 Atividades extra-plano realizadas

- Participação do "IX Simpósio de Computação Tolerante à Falhas e Workshop de Segurança de Sistemas Computacionais", realizado pela SBC (Sociedade Brasileira de Computação) no período de 5 a 7 de Março de 2001, em Florianópolis, SC;

- Maio/2001 – submissão de artigo *"Detecção Móvel de Intrusões"* à Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação.

- Atuação, desde o início do projeto, como um dos administradores da sub-rede do Laboratório Intermídia do ICMC, USP São Carlos;

7 Considerações finais

Este trabalho apresentou a descrição de implementação do cenário de Identificação de Usuários Anômalos, modelado na primeira fase deste projeto e implementado nesta fase final. A constituição do cenário conseguiu expressar claramente o processo de comunicação entre as camadas do modelo proposto pelo trabalho de Bernardes e mostrou uma aplicação prática em um Sistema de Detecção de Intrusões.

Fazendo uso do conceito e implementações deste cenário, novos agentes podem ser desenvolvidos e inseridos em cada camada, o que vem demonstrar a escalabilidade do sistema proposto. Como exemplo, pode-se inserir novos agentes na camada de verificação, aumentando seu grau de especialização, sem a necessidade de alteração nos agentes das camadas superior. O mesmo pode acontecer nas demais camadas.

Algumas dificuldades de testes foram encontradas devido a restrições de abertura de código por parte dos desenvolvedores das plataformas de agentes móveis, já que há pouco tempo a metodologia de trabalho com código aberto está sendo adotada nesta área específica. A não compatibilidade da versão inicial do sistema com Java 2 também não permitiu o uso de novos recursos de segurança desta linguagem e é um dos objetivos que serão destacados no período de renovação desta pesquisa.

8 Solicitação de renovação de bolsa

Solicito a renovação da Bolsa de Pesquisa em Iniciação Científica, para cumprimento das seguintes tarefas:

- Atualização do cenário implementado para a linguagem Java 2, utilizando novos conceitos na evolução da linguagem e do ASDK 1.2;

- Modelagem, implementação e testes de novos agentes nas camadas de notificação e reação no cenário proposto. São previstos agentes de notificação dos usuários através de comunicação por celular e reação de ativação de recursos de *logging* ao invés de simples quebra de conexão e visando coleta de informações para identificação do invasor;

- Submissão de artigo para o Simpósio de Segurança em Informática, no Instituto Tecnológico da Aeronáutica, a ser realizado no período de 24 a 26 de Outubro de 2001;

- Pretende-se também a definição de linhas gerais para métodos de *Análise Forense Básica em Sistemas Linux*, visto que em sistemas que não possuem recursos de segurança como firewall, política de segurança e sistemas detectores de intrusões são comprometidos por invasores e informações vitais para localização dos invasores são perdidas por falta de uma análise do sistema após uma invasão.

Segue o cronograma definido para a nova etapa do projeto:

	Agosto/2001	Setembro/2001	Outubro/2001	Novembro/2001	Dezembro/2001	Janeiro/2002
1. Atualização dos agentes para Java 2						
2. Modelagem e implementação de um novo agente na camada de notificação						
3. Modelagem e implementação de um novo agente na camada de reação						
4. Testes e validação do ASDK em Java 2						
5. Submissão de Artigos para eventos e revistas técnicas						

9 Referências

(ASDK, 2001) IBM ASDK (*Aglets Software Development Kit*). *The Aglets Portal*. Disponível on-line em http://www.aglets.org Visitado em 17/05/2001.

(Bernardes, 1999) BERNARDES, M.C & MOREIRA, E. S. *Avaliação do Uso de Agentes Móveis em Segurança Computacion*al. Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 1999.

(CSI, 2001) *Computer Crime and Security Survey.* Computer Security Institute/Federal Bureau of Investigation. 2001. http://www.gocsi.com

(Figueiredo, 2000) FIGUEIREDO, T.C & SANTANA, M.J. *Interface AMIGO-MPI: Uma abordagem flexível e dinâmica para escalonamento de processos.* Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, Outubro 2000.

(IKV, 2000) IKV++. *Grasshopper Sit*e. Disponível on-line em http://www.grasshopper.de/. Visitado em 9/05/2001.

(JDK, 2001) JDK 1.1 (*Java Development Kit*). Disponível on-line em http://www.javasoft.com/products/jdk/1.1/ Visitado em 17/05/2001.

(Karjoth, 1997) Karjoth, G.; Lange D.; Oshima M. *A Security Model for Aglets.* IEEE Internet Computing Magazine, pág. 68. 1997.

(Lange & Oshima, 1998) Lange, D.B; Oshima, M. Programming and *Deploying Java Mobile Agents with Aglets*. Addison Wesley Longman, Inc. 1998

(Mitsubishi, 2001) MITSUBISHI Eletric Information Center. *Concordia – Java Mobile Agent Technology*. Disponível on-line em http://www.meitca.com/HSL/Projects/Concordia/Welcome.html .Visitado em 9/05/2001.

(Módulo, 2000) *6ª. Pesquisa Nacional de Segurança da Informação.* Módulo Security Solutions S.A., 2000. http://www.modulo.com.br

(Pereira Filho, 2001) PEREIRA FILHO, S. *Avaliação de Ambientes Servidores para Agentes Móveis.* Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, Abril 2001.

(Tryllian, 2000) TRYLLIAN, B.V. *Tryllian White Paper.* http://www.tryllian.com/sub_downl/Tryllian_white_paper.pdf, 2000. Visitada em 9/05/2001.

São Carlos, 1º. de Junho de 2001.

Francisco Gomes Milagres	Prof. Dr. Edson dos Santos Moreira
Bolsista	*Orientador*