UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TATIANA CUSTÓDIO

IMPLEMENTAÇÃO DE UMA APLICAÇÃO DE AGENTES MÓVEIS PARA AVALIAR O IMPACTO DA MOBILIDADE NA SEGURANÇA

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

JOÃO BOSCO MANGUEIRA SOBRAL

Florianópolis, março de 2002.

IMPLEMENTAÇÃO DE UMA APLICAÇÃO DE AGENTES MÓVEIS PARA AVALIAR O IMPACTO DA MOBILIDADE NA SEGURANÇA

TATIANA CUSTÓDIO

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

_____

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

Banca Examinadora

_____

Prof. João Bosco Mangueira Sobral, Dr.

_____

Prof. Luiz Carlos Zancanella, Dr.

_____

Profª. Mirela Secchi Moreti Annoni Notare, Drª.

_____

Prof. Rosvelter João Coelho da Costa, Dr.

“Assim eu quereria o meu último poema.

Que fosse terno dizendo as coisas mais simples e menos intencionais

Que fosse ardente como um soluço sem lágrimas

Que tivesse a beleza das flores quase sem perfume

A pureza da chama em que se consomem os diamantes mais límpidos

A paixão dos suicidas que se matam sem explicação.”

(Manuel Bandeira - O último poema)

À minha mãe Maria Efigênia
Custódio que agora,
aos 54 anos, está começando
seu primeiro curso de
graduação.

Agradecimentos

Gostaria de agradecer a todos aqueles que de alguma forma colaboraram com a realização deste trabalho, seja no ponto de vista técnico ou emocional, àqueles que tiveram uma palavra amiga no momento necessário.

Aos professores que auxiliaram na pesquisa e na implementação. Aos amigos de mestrado, familiares e especialmente a Deus, que sabe que tudo tem seu tempo e que há a hora de plantar e a hora de colher.

Em especial, agradeço a orientação do professor João Bosco Mangueira Sobral.

E, não podia deixar de agradecer a paciência de meu “namorido” Jorge assim como, as palavras de auxílio e orações de minha tia Cida e de meu irmão Márcio.

Sumário

Lista de tabelas

Lista de figuras

Lista de siglas

ACL ..... *Access Control Lists*
API ..... *Application Programming Interface*
ASDK ..... *Aglet Development Kit*
ATP ..... *Aglet Transfer Protocol*
CORBA ..... *Common Object Request Brokers*
CPU ..... *Central Processing Unit*
DCOM ..... *Distributed Component Object Model*
DES ..... *Data Encriyptation Standart*
DOS ..... *Denial of Service*
EJB ..... *Enterprise JavaBean*
HTTP ..... *HyperText Transfer Protocol*
IETF ..... *Internet Engineering Task Force*
IIOP ..... *Internet Inter-ORB Protocol*
IPX ..... *Internetwork Packet Exchange*
JNDI ..... *Java Naming and Directory Interface*
JSP ..... *Java Servlet Pages*
JTA ..... *Java Transaction API*
JVM ..... *Java Virtual Machine*
MAC ..... *Message Authentication Code*
MEITCA ..... *Mitsubishi Eletric Information Tecnology Center América*
ORB ..... *Object Request Broker*
OMG ..... *Object Management Group*
OTS ..... *Object Transaction Service*
RMI ..... *Remote Method Invocation*
RPC ..... *Remote Procedure Calling*
SSL ..... *Secure Socket Layers*
TCP ..... *Transmission Control Protocol*
UDP ..... *User Datagram Protocol*
URL ..... *Uniform Resource Locator*

Apresentação

Esta dissertação é o resultado de vários anos de estudo na área de sistemas distribuídos. Tais estudos ocorreram durante o período compreendido entre os anos de 1997 e 2002. Em um primeiro momento, os estudos estavam voltados a utilização de CORBA e a partir de 1999 o foco voltou-se para a utilização da tecnologia dos agentes móveis.

Este trabalho situa-se na área de concentração de Engenharia de Sistemas Distribuídos deste curso, envolvendo ainda questões relativas à segurança (Segurança e Comércio Eletrônico). Assim, é proposto um modelo para o desenvolvimento de aplicações baseadas em agentes móveis considerando uma política de segurança. O intuito de deste trabalho é reforçar a segurança, pois somente assim, o sucesso comercial na utilização de agentes móveis será atingido.

Este trabalho considera de grande importância não só a proposta de um modelo de desenvolvimento de aplicações baseadas em agentes moveis e a política de segurança a ser adotado, mas também a sua aplicação em um caso real. O modelo é aplicado em plataformas de agentes móveis da IBM denominada *Aglets*.

Resumo

Neste trabalho apresenta-se um modelo para o desenvolvimento de uma aplicação baseada em agentes móveis. Esta aplicação é desenvolvida de acordo com uma política de segurança. Tal política faz com que os riscos existentes na utilização de agentes sejam reduzidos. Questões como a proteção do agente, do computador hospedeiro e da rede são abordados neste trabalho, assim como os tipos de ataque que podem ser encontrados neste paradigma. A identidade é o ponto de partida do modelo de segurança pois, através dela, os elementos podem ser autenticados.

Além das identidades, o modelo de segurança é baseado ainda em permissões associadas a elementos e a recursos do sistema, proteções e autorizações. O modelo de segurança proposto por Karjoth [KARJOTH at al 97], utilizado como base deste trabalho, é aplicado em um caso prático, usando a plataforma de agentes móveis da IBM cujo nome é *Aglets*. Esta plataforma foi desenvolvida utilizando a linguagem Java.

Abstract

This work presents a model for development of an application based on mobile agents. This application is developed according to a security policy. This security policy permits the reduction of the risks in the use of mobile agents. Subjects such as agents, host and net protection are discussed in this work as well the types of attack which may be found in this paradigm. The principals are the starting point of the security model because by them the entities can be authenticated.

Besides the identities, the security model is still based on permissions linked to elements, resources, protections and authorization of the system. The security model proposed by Karjoth [KARJOTH at al 97], used as base of this work, is applied in a practical case, using the IBM platform of mobile agents, called Aglets . This platform was developed by using Java language.

Palavras-chave

Modelo de desenvolvimento de aplicação
Modelo de segurança,
Política de segurança,
Arquitetura de Segurança,
Aplicação Segura,
Agentes móveis, e
Riscos à segurança.

1 Introdução

Este trabalho versa sobre um **modelo** para o desenvolvimento e execução de aplicações baseadas em **agentes móveis**, considerando o **modelo de segurança** adotado, de forma que o mesmo suporte uma definição da arquitetura de **políticas de segurança**.

Agentes móveis oferecem um paradigma para a computação distribuída, mas além de seus benefícios devem ser consideradas as ameaças reais de segurança. Essas ameaças se originam, não apenas através de agentes maliciosos, como também pelos computadores hospedeiros desses agentes. O sucesso comercial de **agentes inteligentes** está largamente ligado à solução de **problemas de segurança** associados ao uso de agentes.

As questões concretas na área de segurança dependem muito do projeto de um agente e de todo o contexto da aplicação em que o mesmo está envolvido. Durante o processo de análise e desenvolvimento define-se um agente como estacionário ou móvel. Ao passo que agentes estacionários estão relacionados a **riscos de segurança gerais** de **aplicações baseadas em rede**, o uso de agentes móveis introduz uma nova dimensão quanto a solução de problemas de segurança. A capacidade de se mover dentro de uma rede significa que ele possui um potencial de perigo maior fazendo com que **novas questões de segurança** precisem ser analisadas.

Os riscos no uso de agentes móveis fazem aparecer novos problemas, tais como:

- Dificuldades a proteção do agente
  - Um computador hospedeiro ameaça um agente;
  - Um agente ameaça outro agente;
  - Uma terceira entidade não autorizada ameaça o agente: neste caso esta entidade pode tentar alterar mensagens trocadas entre computadores hospedeiros ou revelar seu conteúdo;

- Dificuldades na proteção do computador hospedeiro
  - Um agente visitante da máquina hospedeira ataca o *host*;
  - Uma terceira entidade não autorizada ataca a máquina hospedeira: esta entidade pode enviar um grande número de agentes para o *host* de forma a sobrecarregá-lo;
- Dificuldades na proteção da rede
  - agente ameaça a rede: neste caso um agente pode se multiplicar de forma a sobrecarregar toda a rede;

Atualmente existem diversas plataformas objetivando um ambiente para o desenvolvimento e execução de agentes móveis. As plataformas que surgiram inicialmente eram baseadas em TCL. Com a evolução da linguagem Java, novas plataformas foram apresentadas ao mercado tendo como base esta linguagem.

Este trabalho aborda os fatores necessários para a criação de uma aplicação baseada em agentes móveis que atenda as necessidades do usuário levando em consideração os seguintes fatores: a existência de um **modelo de segurança genérico** que possa ser suportado por plataformas de agentes móveis baseadas em Java e o uso deste mesmo **modelo de segurança** para a proteção da aplicação e dos sistemas local e remoto.

Como instrumento que permite a utilização desse modelo em um sistema real, é utilizada a plataforma de agentes móveis que define e executa os *aglets*. *Aglets* são agentes móveis Java que auxiliam pessoas e agem em nome delas.

Como histórico da pesquisa já realizada, cita-se os seguintes trabalhos:

- Em [KARJOTH at al 97], "Um modelo de segurança para *aglets*" é apresentado com situações em que a segurança dos agentes é considerada vulnerável na plataforma de agentes da IBM, *Aglets*.

- Em [SANDER & TSHUDIN 97], é apresentado o trabalho "Protegendo Agentes de *Hosts* Maliciosos" em que a proteção de agentes é focada no ataque dos *hosts* em que o agente trafega.

- "Segurança e Confiabilidade do *Concordia*" [WALSH at al 98], é um trabalho que apresenta a estrutura da plataforma *Concordia* da Mitsubishi e de que forma aspectos de segurança foram abordados no desenvolvimento do projeto.

- Em [VALDO 00], um sistema para integrar comércio eletrônico e ERP através de agentes móveis é apresentado, onde uma aplicação de comércio eletrônico foi desenvolvida fazendo uso de agentes móveis.

Tais trabalhos, embora tenham dado uma contribuição precursora, não apresentaram soluções completas para problemas de segurança associados aos agentes móveis, como os aqui apontados. Aspectos como a ausência de questões que envolvam segurança em um trabalho voltado ao comércio eletrônico e o fato de os modelos de segurança não terem sidos usados em uma aplicação prática justificam tal afirmação.

Este trabalho é baseado em "Um modelo de segurança para *Aglets*", [KARJOTH at al 97] e sua contribuição é a associação de um modelo de desenvolvimento de aplicações ao modelo de segurança proposto por este autor.

O trabalho presente está estruturado como segue: no capítulo 2 são abordados os conceitos de agentes móveis. No capítulo 3 são apresentadas características particulares de plataformas de agentes móveis desenvolvidas utilizando a linguagem Java. No capítulo 4 os aspectos de segurança. No capítulo 5, as características pertinentes ao modelo de segurança, a linguagem de definição de uma política de segurança, além do modelo proposto por Karjoth. No capítulo 6 as características do modelo de segurança apresentados anteriormente serão implementados em um caso prático usando a plataforma dos *aglets,* da IBM e finalmente, no capítulo 7 são apresentadas as conclusões do trabalho.

2 Tecnologia de Agentes Móveis

2.1 Caracterização de um agente de software

Não há uma definição universalmente aceita para agentes. Essencialmente, este tema é tão vasto que existem várias definições para um mesmo termo. São apresentadas a seguir algumas daquelas que são consideradas importantes por terem sido feitas por pesquisadores que se destacaram na da área dos agentes.

"*Agentes são programas de computadores que empregam técnicas de inteligência artificial para fornecer assistência ativa para usuários em tarefas baseadas no computador.*" [MAES 94].

"*Uma entidade de software com um programa contido em si próprio que tem a capacidade de controle de sua própria criação e ação, baseado na percepção de seu ambiente, e com isso persegue um ou mais objetivos.*" [JENNINGS 96].

"*Um agente artificial que opera em um ambiente de software.*" [CROFT 97].

"*Agentes de software são programas que ajudam pessoas e atuam ao seu lado. A função dos agentes é permitir que pessoas deleguem tarefas a eles.*" [LANGE & OSHIMA 98].

"*Um programa que pode migrar através de nós de uma rede carregando o seu estado de execução nos horários e para locais de sua própria escolha*". [KOTZ & GRAY 99].

Baseado em estudos realizados, não existe uma definição única e correta, pois em todas elas encontram-se peculiaridades que as tornam de suma importância no contexto de agentes, apesar de estarem focadas nas áreas as quais os pesquisadores estão associados. Uma definição complementa a outra, de forma a mostrar como o universo associado à tecnologia de agentes pode ser abrangente. Como o foco deste trabalho está associado a agentes móveis o conceito adotado a partir deste momento será o descrito por Lange. Assim sendo, pela perspectiva do usuário final, um agente de software é um programa que o ajuda a realizar um determinado serviço ou receber e executar tarefas que lhe são delegadas; e para o sistema, o agente de software é um objeto que está situado em um ambiente de execução e possui as seguintes propriedades:

- **Reatividade**: É capaz de sentir alterações no ambiente em que está e, baseado nestas alterações, executar determinadas ações.

- **Autonomia**: Possui o controle das ações que ele mesmo executa.

- **Orientação a objetivos**: É capaz de agir de forma pró-ativa, tentando assim, alcançar o objetivo previamente delegado a ele.

- **Continuidade de execução**: Permanece em execução a não ser que tenha alcançado o seu objetivo ou tenha recebido uma ordem para finalizar seu processamento.

- **Mobilidade**: Habilidade de trafegar através da rede, fazendo uso de um itinerário ou destino previamente definido.

- **Comunicabilidade**: Habilidade de se comunicar com outros agentes existentes no sistema.

- **Aprendizagem**: Baseado em experiências passadas, um agente é capaz de saber qual sua atitude diante de um fato.

Quanto à mobilidade, podemos descrevê-la como sendo uma propriedade complementar de agentes pois um agente pode ficar localizado em uma determinada estação e lá permanecer durante todo o seu período de vida – agente estacionário – ou utilizar-se das demais máquinas da rede para executar suas tarefas – agente móvel. Dessa forma, podemos definir agentes quanto a mobilidade:

- **Agentes estacionários**: são agentes que executam apenas no sistema em que foram criados. Caso necessitem interagir com outros agentes ou precisem de informações localizadas fora de seu sistema de origem, fazem uso de um mecanismo de mensagens como RPC.

- **Agentes móveis**: não estão limitados ao local onde foram criados. Possuem a habilidade de transportar-se de um sistema para outro. Esta capacidade é o que possibilita ao agente móvel alcançar o sistema que contenha o objeto com o qual este deseja interagir e, por estar na mesma estação em que o objeto está, obter privilégios que não seriam dados a um elemento externo ao sistema.

2.2 Características importantes dos agentes móveis

Agentes móveis apresentam as seguintes vantagens:

- **Redução do tráfego na rede**: sistemas distribuídos freqüentemente trabalham com protocolos que envolvem múltiplas interações para o cumprimento de uma determinada tarefa, com agentes isto não ocorre pois eles são enviados para a estação destino e lá executam suas tarefas. Dessa forma, ao invés de serem enviados dados ao processamento, o próprio processamento se desloca ao local onde estão os dados.

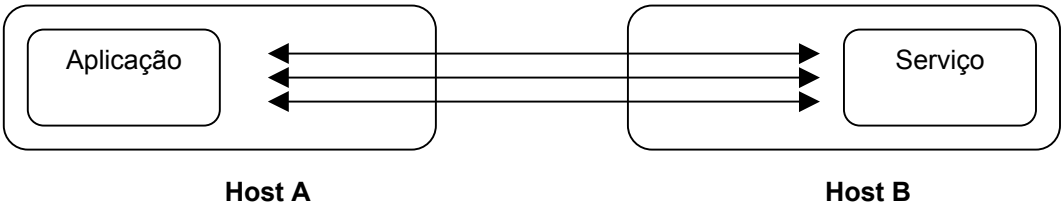

Figura 1 - Comunicação utilizando RPC

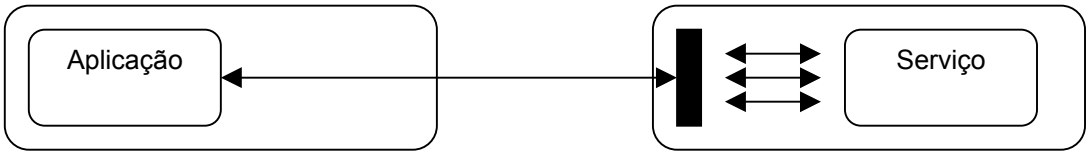

Figura 2 - Comunicação utilizando agentes móveis

- **Redução do uso de recursos no servidor**: agentes móveis usam as capacidades e recursos do computador hospedeiro para processar suas tarefas. O usuário do agente móvel pode detectar uma redução significante do uso de seu próprio sistema. Por outro lado, a capacidade utilizada no *host* remoto será acrescida correspondentemente porque, além de suas próprias tarefas, o computador hospedeiro disponibilizará recursos para o processamento da tarefa do agente presente em seu sistema.

- **Diminuição do atraso na rede**: sistemas considerados críticos precisam de respostas em tempo real para alterações ocorridas no ambiente de execução. Atrasos são inaceitáveis neste tipo de sistema. Agentes móveis representam a solução adequada para este problema pois estes são enviados do computador central às demais máquinas da rede de forma a executar localmente o seu controle, causando assim um atraso completamente nulo.

- **Execução assíncrona e autônoma**: freqüentemente a comunicação entre dispositivos móveis baseia-se em conexões de rede muito frágeis. Algumas tarefas necessitam de interação contínua e devido a esta fragilidade tornam-se impossíveis de serem realizadas. Para solucionar este problema, as tarefas podem ser embutidas nos agentes móveis que são enviados através da rede. Depois que o agente é transmitido, ele torna-se independente do processo que o

criou e assim pode operar de forma assíncrona e autônoma. O sistema pode reconectar-se após um determinado tempo e coletar o agente e o resultado da tarefa que lhe havia sido delegada.

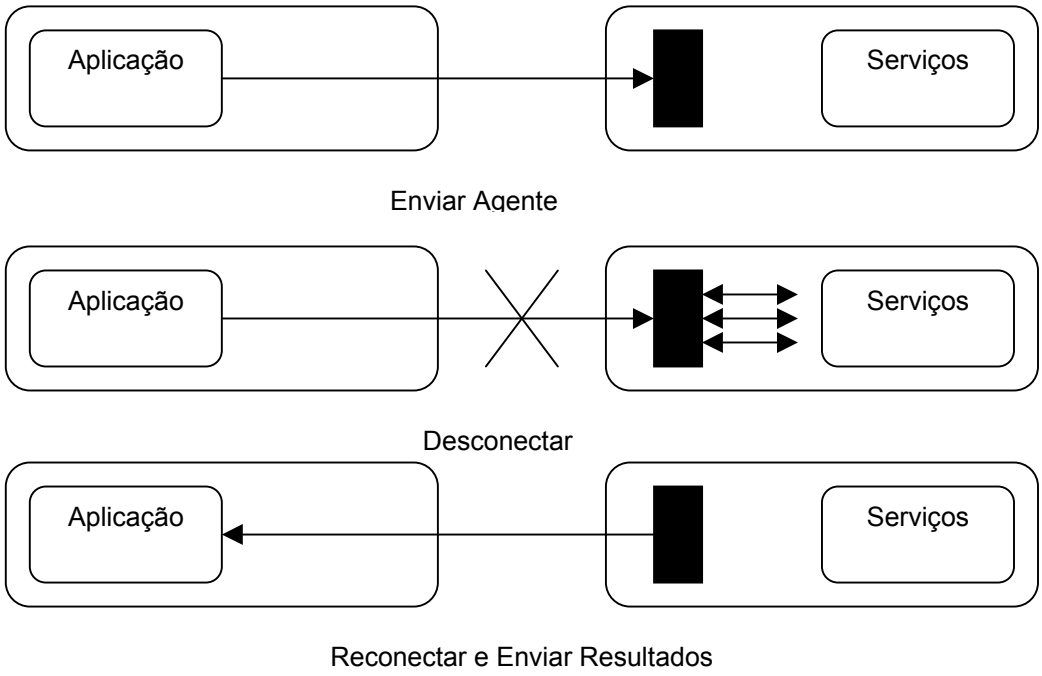

Figura 3 - Execução assíncrona em agentes móveis

- **Estrutura descentralizada**: agentes móveis suportam o paradigma clássico do cliente-servidor. Entretanto, sua mobilidade permite a criação de estruturas descentralizadas em um nível maior do que as dos programas tradicionais. Diferente das RPCs, o processamento não é distribuído apenas em dois computadores, mas em tantos quanto forem necessários. Se há um sistema de gerenciamento inteligente de rede, qualquer sobrecarga nos nodos podem ser percebidos, o agente pode fazer uso dessa informação e endereçar as tarefas àqueles nodos que estão com a menor carga de trabalho no momento.

- **Adaptação dinâmica**: agentes móveis têm a habilidade de sentir o ambiente no qual estão executando e reagir de forma automática às mudanças. Caso haja urgência em uma determinada tarefa, o agente pode multiplicar-se entre as estações da rede de forma a manter a configuração ótima e solucionar rapidamente o problema.

- **Heterogeneidade**: uma rede de computadores é fundamentalmente heterogênea (tanto em *hardware* quanto em *software*). Agentes móveis são geralmente independentes da camada de transporte e do computador; e dependentes somente de seu ambiente de execução, o que fornece condições ótimas para a integração de sistemas não compatíveis.

- **Robustez e tolerância à falhas**: A capacidade que os agentes móveis possuem de reagir dinamicamente a situações e eventos desfavoráveis os tornam fortes candidatos à construção de sistemas distribuídos robustos e tolerantes a falhas.

2.3 Paradigmas da computação em rede

Agentes móveis propiciam um paradigma poderoso e uniforme no conceito de redes. Os agentes podem revolucionar o projeto e o desenvolvimento de sistemas distribuídos e para comprovar isso, uma comparação entre os seguintes paradigmas será feita:

- **Cliente-servidor***:* no paradigma cliente-servidor, um servidor oferece um conjunto de serviços. O código que implementa estes serviços se localiza no próprio servidor. Como o servidor é aquele que possui o código que implementa o serviço diz-se que ele possui o *know-how*. E como todo o processamento também é feito nele, adota ainda os papéis de processador e de fornecedor de recursos. Muitos sistemas distribuídos foram criados com base neste paradigma como o RPC, CORBA e RMI.

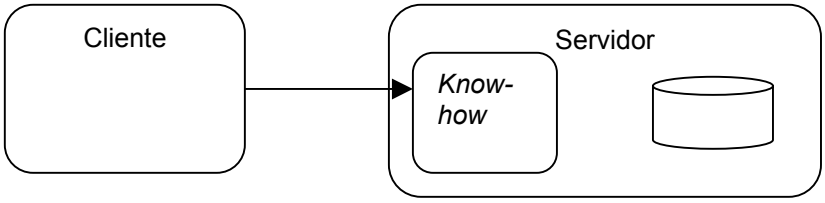

Figura 4 - Paradigma cliente-servidor

- **Código sob demanda**: no paradigma de código sob demanda o usuário precisa obter a aplicação (*know-how*) quando precisar. Apenas o código é recebido pelo cliente. O cliente utiliza as capacidades de seu computador, assim como os seus próprios recursos para executar sua aplicação. Diferentemente do paradigma

cliente-servidor, o usuário não necessita pré-instalar nada em sua máquina, pois tudo aquilo que for necessário para a execução da aplicação será trazida durante a execução. *Applets* Java e *servlets* são excelentes exemplos deste paradigma. *Applets* são trazidas da *web* pelo navegador e executadas localmente enquanto os *servlets* são enviados para um servidor *web* remoto e lá executam.

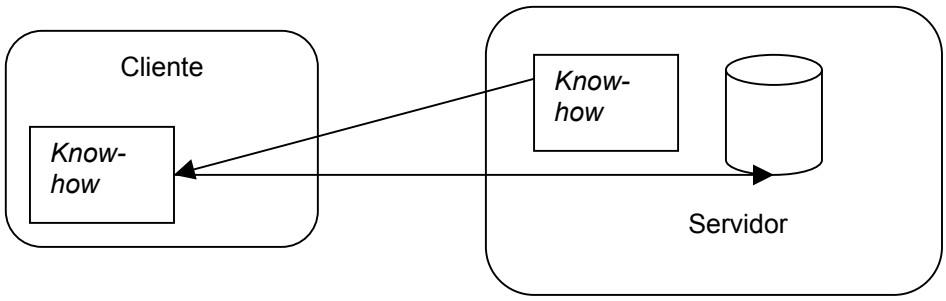

Figura 5 - Paradigma código sob demanda

– **Agentes móveis**: a característica chave do paradigma de agentes móveis é que a qualquer *host* da rede é permitido um alto grau de flexibilidade pois ele pode assumir a posição de fornecedor de aplicação(*know-how*), recursos e processamento. Essas capacidades podem estar associadas a existência de recursos locais para satisfazê-las.

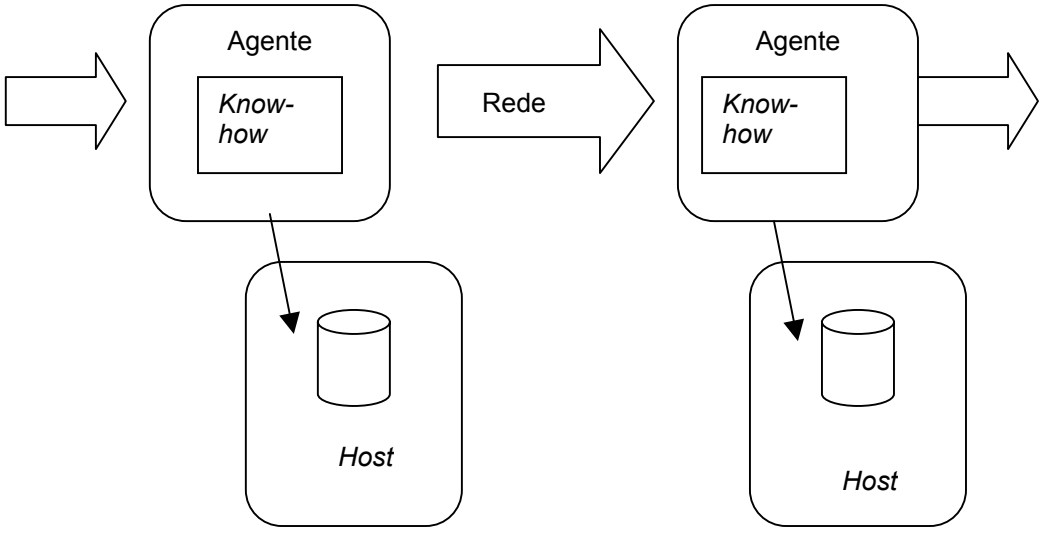

Figura 6 - Paradigma de agentes móveis

Se o paradigma de agentes móveis for comparado aos demais veremos que ele tende a um alto nível de flexibilidade. O cliente e o servidor se fundem e tornando-se um *host*.

2.4 Aplicações em que a utilização de agentes móveis é indicada

Há várias aplicações em que pode-se fazer uso da tecnologia dos agentes móveis, dentre elas pode-se citar:

- **Comércio eletrônico**: agentes móveis são bem adaptados ao comércio eletrônico. Uma transação comercial pode requerer acesso em tempo real à recursos remotos como itens existentes no estoque, por exemplo. Diferentes agentes podem ter diferentes objetivos e podem implementar diferentes estratégias para alcançar estes objetivos. Os agentes incorporam as intenções de quem os criou e agem e negociam em nome deles.

- **Assistência pessoal**: a habilidade que os agentes móveis possuem de se locomover entre *hosts* remotos os torna capazes de executar tarefas em nome de seu criador sem a interferência do mesmo. O assistente remoto irá operar independentemente dos limites da rede e sem que o computador de seu criador tenha que estar conectado. Tarefas como agendar reuniões com um grupo extenso de pessoas podem ser feitas por ele. Este elemento irá interagir com os demais agentes que representam outros membros e, baseado na agenda de cada um deles, localizar uma data considerada ideal para todo o grupo.

- **Recuperação de informações distribuídas**: este é um exemplo popular de aplicações indicadas à agentes móveis. Ao invés de mover uma grande quantidade de dados para um mecanismo de busca, pode-se enviar agentes diretamente às fontes de informações, onde estes possam criar localmente índices para um posterior reenvio do agente e dos dados à sua origem.

- **Monitoramento e notificação:** Esta aplicação clássica salienta a natureza assíncrona do agentes móveis. Agentes podem esperar até que uma determinada

informação se torne disponível retornando ao local de origem apenas quando possuir tal dado.

- **Disseminação de informações:** Agentes móveis englobam o chamado modelo *push* da internet. Eles podem disseminar informações como a atualização de *software* para usuários. O agente traz os componentes do novo *software* assim como os procedimentos necessários para a sua instalação e de forma autônoma atualizar e gerenciar o *software* no computador.

- **Processamento paralelo:** Dado que agentes móveis podem criar uma cascata de clones de si mesmos na rede, um uso potencial da tecnologia de agentes móveis é administrar o processamento paralelo de tarefas.

2.5 Sistemas de agentes móveis

2.5.1 Java

- ***Aglets*:** *Aglet* da IBM é uma plataforma de agentes móveis que suporta o conceito de autonomia de execução e roteamento dinâmico de itinerário. Pode-se pensar no *aglet* como sendo uma generalização e extensão dos *applets* e *servlets* Java. Os *aglets* são armazenados por um servidor *Aglet* assim como os *applets* são no servidor *Web*. O servidor *Aglet* fornece um ambiente para os *aglets* executarem e a máquina virtual Java (JVM). O gerente de segurança *Aglet* torna seguro o recebimento e hospedagem do A*glet*. O termo A*glet* (*lightweight agent*) é uma combinação de agente e *applet* (*lightweight application*).

- ***Concordia*:** O *Concordia* da Mitsubishi é uma plataforma para o desenvolvimento e gerenciamento de aplicações de agentes móveis extensível a qualquer sistema que suporte Java. *Concordia* consiste em múltiplos componentes, todos escritos em Java, que são combinados de forma a propiciar um ambiente completo para aplicações distribuídas. Um sistema *Concordia*, de forma simplista, é composto por uma JVM, um servidor e uma série de agentes.

- ***Voyager:*** *Voyager* da *ObjectSpace* é uma plataforma para computação distribuída acrescida de agentes Java. *Voyager* fornece um extenso conjunto de objetos com capacidade de enviar e receber mensagens, também permite que os objetos se movam como agentes em uma rede. *Voyager* combina as propriedades de um ORB (*Object Request Broker*) baseado em Java, e um sistema de agentes móveis. Dessa forma permite que os programadores Java criem aplicações em rede usando ambas as técnicas: a tradicional e a programação distribuída acrescida de agentes.

2.5.1.1 Características

Como Java é orientada a objetos e amplamente utilizada em rede, torna-se a linguagem ideal para a programação de agentes móveis. Uma série de características mostram porque Java é indicada quando se deseja trabalhar com o paradigma dos agentes móveis. Mais adiante serão apresentados aspectos impeditivos a sua utilização.

- **Benefícios**
  - **Independente de plataforma:** A linguagem Java é projetada para operar em sistemas heterogêneos. Para fazer com que uma aplicação Java possa executar em qualquer lugar da rede, o compilador gera *bytecodes* que no momento de sua execução serão interpretados pela máquina virtual Java. Para este código ser gerado em um determinado computador, o sistema *runtime* do Java deve estar presente. Até mesmo suas bibliotecas são independentes de plataforma, pois são escritas em Java.

  - **Execução segura:** Java é projetado para a utilização na Internet e intranets. A demanda por segurança influenciou seu projeto de diversas maneiras. Como exemplo pode-se citar os ponteiros do Java que não permitem a sobrescrita de memória e corrupção de dados.

  - **Não permite *type casting*:** Os programas não conseguem forjar acesso a dados privados em objetos que eles não tem permissão, fator este que impede a execução de determinados tipos de vírus. Caso um *bytecode* seja alterado, o sistema *runtime* do Java garante que o código não estará apto a violar a

semântica básica da linguagem Java. Possui um gerente de segurança para checar todas as operações potencialmente inseguras como: acesso a arquivos, conexões de rede, entre outras.

- **Leitura dinâmica de classes**: Este mecanismo permite à máquina virtual carregar e definir classes em tempo de execução. Fornece um espaço de nome protegido para cada agente, permitindo a estes executarem de forma segura e independentes uns dos outros. O mecanismo de leitura de classes é extensível e possibilita que as classes sejam carregadas através da rede.

- **Programação *multithread***: Agentes são por definição autônomos. Um agente executa independentemente de outro que resida no mesmo espaço. A possibilidade de execução em seu próprio processo, chamado *thread* de execução, é uma forma de se habilitar agentes a se tornarem autônomos. Além da programação *multithread*, a linguagem Java também suporta um conjunto de primitivas de sincronização de forma a permitir a interação entre os agentes.

- **Serialização de objetos**: Uma característica chave dos agentes móveis é que eles podem ser serializados e deserializados. A linguagem Java convenientemente fornece um mecanismo de serialização embutida que pode representar o estado de um objeto em uma forma serializada, suficientemente detalhada, para que o objeto seja reconstruído mais tarde.

- **Reflexão computacional**: O código Java pode descobrir informações sobre campos, métodos, construtores e carregadores e tais informações podem ser usadas para operar objetos de parceiros subordinados, tudo dentro de restrições de segurança. A reflexão implementa a necessidade dos agentes serem inteligentes sobre si mesmos e outros agentes.

- **Pontos negativos**

Segundo [LANGE & OSHIMA 98], apesar da linguagem Java ser altamente indicada para a criação de agentes móveis é necessário que seja dada a devida atenção a determinadas falhas. Algumas delas podem ser ajustados de forma a não prejudicar o

sistema, outras contudo, são muito sérias e apresentam implicações para todo o projeto conceitual e desenvolvimento de agentes móveis usando Java.

- **Suporte inadequado ao controle de recursos:** A linguagem Java não fornece uma maneira de controlar a quantidade de recursos do sistema que é consumida por um determinado objeto. Por exemplo, um agente pode executar vários *loops* de forma a desperdiçar o uso do processador e aumentar o consumo de memória. Estas ações podem acarretar num ataque na segurança do sistema conhecido como DOS. Neste ataque o agente se multiplica no computador e esgota seus recursos de modo a tornar impossível ao operador controlar a máquina.

- **Não possui suporte para a preservação e retomada do estado de execução:** No momento é impossível em Java recuperar o estado de execução completo de um dado objeto. Informações como o estado do contador do programa e a pilha de execução são permanentemente esquecidos nos programas Java. Para um agente móvel reiniciar propriamente a sua execução em um *host* remoto ele deve confiar nos valores de seus atributos internos e eventos externos.

2.5.2 Não Java

- ***Agent Tcl:*** *Agent Tcl* do Darthmouth College é um sistema de agentes móveis escrito em Tcl. Possui navegação extensiva, serviços de comunicação, mecanismos de segurança e ferramentas de depuração e rastreamento. O principal elemento do *Agent Tcl* é um servidor que executa em cada máquina e permite que informações do estado do agente sejam armazenadas. Quando um agente deseja migrar para uma nova máquina, ele chama a função *agent_jump*, que automaticamente captura o estado completo do agente e envia estas informações para o servidor destino.

- **Ara:** Ara, baseado em Tcl, da Universidade de Kaiserlautern é uma plataforma para execução portável e segura de agentes móveis em redes heterogêneas. O projeto de pesquisa foi originalmente concebido com o intuito de suportar agentes móveis que assegurassem segurança e execução portável e pouco focado em padrões de cooperação, ambiente inteligente e modelagem de usuários.

- **TACOMA:** O projeto das universidades de Tomso e Cornell foca-se no uso de agentes para solucionar problemas usualmente destinados ao sistema operacional. O TACOMA é baseado em UNIX e em TCP. O sistema suporta agentes escritos em C, Tcl/Tk, Perl, Python e Scheme(Elk) e foi implementado usando a linguagem C.

2.6 Padronização de agentes móveis – MASIF

Os sistemas acima descritos diferem em sua arquitetura e implementação, impedindo assim, a interoperabilidade e o desenvolvimento da tecnologia de agentes móveis voltada ao mercado. Para promover esta interoperabilidade, alguns aspectos de agentes móveis necessitam ser padronizados. As empresas *Crystaliz*, *General Magic Inc.*, *GMD Fokus*, IBM e o *Open Group* têm buscado desenvolver uma padronização para agentes móveis(MASIF) e levar à apreciação do OMG.

Claramente o MASIF não se preocupa com interoperabilidade de linguagem. Interoperar a linguagem de criação de objetos móveis é muito difícil, dessa forma MASIF se limita a interoperar sistemas de agentes escritos usando a mesma linguagem mas desenvolvidos por vendedores diferentes.

2.7 Elementos de um agente

Os dois conceitos fundamentais do modelo de agentes móveis são:

2.7.1 Agente

Um agente móvel é uma entidade que possui cinco atributos: estado, implementação, interface, identificador e mandante, onde:

- **Estado** é o conjunto das informações necessárias ao agente quando este precisa retomar sua execução após a transferência de um ponto a outro da rede.

- **Implementação** é o atributo necessário para que o agente possa executar a si mesmo independente da sua localização na rede. Quando um agente trafega pela

rede, duas situações podem ser implementadas: levar todo o código necessário consigo ou viajar até a estação destino e analisar o código existente, recuperando somente o código necessário a execução.

- **Interface** é o atributo necessário para a comunicação do agente.

- **Identificador** é o atributo necessário para reconhecer e localizar agentes em trânsito. Cada agente possui um identificador que é único durante toda sua vida, facilitando assim, sua identificação e localização através de serviços de diretório por exemplo.

- **Proprietário** é o atributo necessário para determinar responsabilidades morais e legais. Um proprietário é uma entidade cuja identidade pode ser autenticada por algum sistema cujo agente esteja tentando acessar. Um proprietário pode ser uma pessoa, uma organização ou uma corporação. Sua identidade pode ser composta por um nome e possivelmente outros atributos.

2.7.2 Local

Um local é o ambiente no qual os agentes operam, ou seja, é o contexto no qual um agente pode executar. Este local é visto como um ponto de entrada para que um agente visitante possa executar seu código. Além disso, o local pode ser caracterizado como o sistema operacional dos agentes.

3 Plataformas de agentes móveis em Java

3.1 *Aglets*

Após o surgimento da linguagem Java, o laboratório de pesquisas da IBM de Tóquio no Japão decidiu desenvolver um ambiente para a programação e execução de agentes móveis denominado *Aglets*, desenvolvido utilizando esta nova linguagem.

Sua primeira versão foi distribuída em 1996 na forma de um *kit*, denominado ASDK, acessível via Internet. Posteriormente outras versões foram liberadas.

O ASDK possui uma interface gráfica de usuário denominada *Tahiti*. Esta interface é considerada a representação gráfica de um contexto, através do qual pode-se criar, despachar, monitorar e extinguir agentes. *Tahiti* habilita ainda o estabelecimento de privilégios do agente no servidor. Os agentes criados no *Tahiti* podem se tornar persistentes, ser clonados e enviar mensagens síncronas e assíncronas. A utilização implica que tanto a origem quanto o destino possuam o *software* instalado.

3.1.1 Elementos Básicos

O modelo dos A*glets* define uma série de elementos:

- ***Aglet:*** é um objeto móvel Java que visita estações habilitadas para receber *Aglets* em sua rede. É autônomo, porque possui sua própria *thread* de execução após chegar a uma estação e é reativo porque responde às mensagens recebidas.

- ***Proxy:*** um *proxy* é uma representação de um a*glet*. Serve como um escudo, protegendo o a*glet* de acessos diretos a seus métodos públicos. O *proxy* também

fornece transparência de localização para o a*glet*, isto é, esconde a real posição do a*glet* na rede.

- **Contexto:** um contexto é o ambiente de execução dos a*glets*, corresponde ao local definido anteriormente.

- **Identificador**: o identificador está associado a cada a*glet*. Todo a*glet* possui um identificador global único que é inalterado durante toda a sua vida.

Os *aglets* podem executar uma série de operações:

- **Criação** (*creation*): esta operação sempre é executada em um contexto. O novo *aglet* inicia a sua execução assim que sua inicialização for bem sucedida. É uma das formas de trazer o *aglet* à vida.

- **Clonagem** (*cloning*): produz uma cópia idêntica do *aglet*. As únicas diferenças são seu identificador e o fato de a execução ser iniciada no novo *aglet*. Isto deve-se ao fato das *threads* de execução não serem duplicadas. Esta é a segunda forma de trazer um *aglet* a vida.

- **Finalização** (*disposal*): depois de criado, um *aglet* pode ser destruído. A finalização de um *aglet* pára a sua execução e remove o objeto do contexto atual.

- **Emissão** (*dispatching*): a emissão é uma mobilidade ativa realizada pelos *aglets*. O *aglet* empurra a si próprio da estação corrente para uma estação remota.

- **Retração** (*retracting*): os *aglets* também podem realizar uma mobilidade passiva. Consiste em uma estação remota puxar o *aglet* da estação corrente. Movendo um aglet de um contexto para o outro, ele é removido do contexto atual e inserido no contexto de destino, onde sua execução é reiniciada.

- **Ativação/Desativação** (*activation/deactivation*) : a desativação de um *aglet* é a habilidade que este possui de suspender temporariamente a execução de si próprio e armazenar seu estado em um meio secundário. A ativação possibilita a

execução do processo inverso. Isto é útil quando se deseja que um *aglet,* que está consumindo recursos, entre em um estado de hibernação para liberar um determinado recurso e retome sua execução posteriormente.

- **Comunicação** (*messaging*): múltiplos agentes podem trocar informações de acordo com uma determinada tarefa.

3.1.2 O modelo de eventos

O modelo de programação dos *Aglets* é baseado em eventos. Neste modelo existem *listeners* que disparam as devidas ações quando um determinado evento ocorre. Existem três *listeners* dentro do modelo *Aglet*:

- ***Clone listener***: é disparado quando ocorrem eventos de duplicação. Pode-se desta forma programar tarefas antes do *aglet* ser duplicado, quando a cópia é criada e depois da duplicação ter ocorrido.

- ***Mobility listener***: é disparada quando um evento de mobilidade ocorre; com isso pode-se preparar ações antes do *aglet* viajar, antes de ser retraído ou quando chega em um novo contexto (local).

- ***Persistence listener***: espera por eventos de persistência. Permite ao programador executar ações quando um *Aglet* está sendo desativado e depois de ter sido ativado.

3.1.3 O modelo de comunicação

Toda a comunicação dos *Aglets* é por passagem de mensagens. A seguir são exibidos alguns componentes deste modelo de comunicação:

- ***Message***: uma mensagem é um objeto trocado entre *aglets*. Este modelo permite passagem de mensagens síncronas e assíncronas entre *aglets*.

- ***Future reply***: é usado em mensagens enviadas como um *handle* e permite ao emissor da mensagem receber uma resposta assíncrona.

- ***Reply set***: pode conter múltiplos objetos como respostas futuras e é usado para buscar resultados assim que eles estejam disponíveis. É possível também receber uma resposta, ignorando as subseqüentes.

3.1.4 Segurança

É permitida a adoção de políticas de segurança adotadas mediante interface gráfica existente no próprio servidor *Tahiti*.

Pode-se proteger o computador hospedeiro de agentes maliciosos através da autenticação de domínios. Podem ser autenticados o usuário, o *host* e o agente.
A integridade dos agentes durante a transmissão pode ser feita usando criptografia de chave simétrica.

3.2 *Concordia*

Concordia é uma plataforma para desenvolvimento e gerenciamento de agentes móveis escrita em Java. Foi desenvolvido pela *Mitsubishi Eletric Information Tecnology Center América* (MEITCA), com sede em Massachussets/EUA.

O produto em sua versão de avaliação não compreende todas as características do *Concordia*, mas aqui serão descritas as características de sua versão completa.

Concordia objetiva o suporte à colaboração, persistência, transmissão confiável e segurança do agente.

3.2.1 Elementos Básicos

Concordia é constituído de três partes: uma máquina virtual Java, um servidor *Concordia* e, pelo menos um agente. Toda a codificação de componentes é feita em Java.

- **Componentes do Servidor**

O servidor *Concordia* é composto de oito módulos que, instalados e em execução,

lidam com todas as tarefas relacionadas ao manuseio dos agentes. Estes módulos são:

- **Gerenciador de agentes** – *Agent Mananger* – Provê a infra-estrutura de comunicação que permite ao agente, um objeto Java, mover-se pela rede. Compõe o ambiente para criação, execução e destruição do agente. Gerencia o ciclo de vida do agente. Possui como característica o fato de abstrair a interface de rede, não sendo necessário ao programador do agente incluir código sobre as especificidades da rede.

- **Gerenciador de segurança** – *Security Mananger* – Tem como função manter a segurança e a integridade dos agentes móveis e dos recursos da máquina na qual está instalado o servidor *Concordia*. É responsável pela identificação dos usuários, autenticação dos agentes, proteção dos recursos do servidor, segurança e integridade do agente e dos dados que carrega. Autoriza ainda o carregamento dinâmico de classes Java necessárias ao funcionamento do agente. Possui uma interface de usuário para o monitoramento e configuração dos atributos de segurança relativos a estes e aos serviços a serem prestados. O estabelecimento da segurança pode se dar de vários modos: para sistemas altamente seguros pode-se não estabelecer nenhum tipo de credencial de acesso, para redes que utilizam canais públicos como por exemplo a Internet, pode-se utilizar mecanismos mais fortes para a criptografia e autenticação.

- **Gerenciador de Persistência** – *Persistence Mananger* – Mantém um registro do estado dos agentes móveis em trânsito pela rede, caso ocorra algum problema, reinicia o agente do ponto de execução anterior a falha.

- **Gerenciador de Comunicação entre Agentes** – *Inter-Agent Comunication* - Controla o registro de eventos e mensagens de agentes móveis, podendo alcançá-los em qualquer ponto a qualquer momento. Juntamente com o servidor *Concordia* pode distribuir eventos, se necessário, para sincronização e compartilhamento de dados entre agentes ou *multicasting*.

- **Gerenciador de Fila** – *Queue Mananger* - Responsável pela garantia de entrega dos agentes móveis entre servidores *Concordia*, especialmente em redes

não confiáveis. Mantém os agentes enquanto estes aguardam a oportunidade de executar suas tarefas, de acordo com prioridades estabelecidas para o acesso a determinado nó da rede, mantendo seus estados. Refaz solicitações quando o servidor *Concordia* for desconectado da rede por algum motivo.

- **Gerenciador de Diretório** – *Directory Mananger* – Provê o serviço de nomeação para agentes e aplicações, através do qual os agentes localizam os serviços na rede. Pode ser configurado de várias maneiras, de acordo com as necessidades dos serviços prestados.

- **Gerenciador de Administração** – *Administration Mananger* – Permite a administração remota do *Concordia*. Este módulo permite o gerenciamento simultâneo de muitos servidores *Concordia* através de uma interface com o usuário. O módulo de administração do *Concordia* é responsável pelo gerenciamento de todos os serviços, tais como: gerenciamento de agentes (*Agent Mananger*), gerenciamento da segurança (*Security Mananger*) e gerenciador de eventos (*Event Mananger*).

- **Biblioteca de Ferramentas para Agentes** – *Agent Tool Library* – Conjunto de APIs (*Application Program Interface*) do *Concordia* para administração, transporte de agentes e alterações de aplicativos associados. Inclui todas as classes necessárias ao desenvolvimento de agentes móveis, inclusive a classe *Agent* e outras derivadas de classes do Java, com interfaces para o *Concordia*.

3.2.2 Segurança

- **Proteção do Agente**

  Para a proteção do agente durante sua transmissão *Concordia* oferece três opções:
  - a utilização do protocolo de segurança SSL3;
  - a utilização de outro protocolo estabelecido pelo usuário;
  - não usar nenhum protocolo de segurança.

Por outro lado, proteções de armazenamento permitem que o agente não seja acessado nem modificado durante seu armazenamento. Este tipo de proteção é essencial quando o *Concordia* não está sendo executado em um sistema operacional seguro. Os *bytecodes* do agente armazenado são encriptados usando algoritmo de encriptação de chave simétrica. *Concordia* usa a extensão de criptografia do Java em que se pode fazer uso dos seguintes algoritmos simétricos: IDEA, DES, RC4, RC5, Misty ou DES triplo.

- **Proteção de Recursos**

Faz uso do modelo de caixa de areia utilizado pela linguagem Java. Neste modelo o acesso a recursos é definido de acordo com o grau de confiabilidade no usuário.

3.3 *Voyager*

O *Voyager* foi elaborado pela *ObjectSpace*, empresa fundada em 1992 e localizada em Dallas.

Voyager é uma plataforma para o desenvolvimento de aplicações distribuídas, baseada em Java, que possibilita a execução de agentes móveis. Possui ainda a capacidade de criar e mover objetos remotamente, através da utilização de mensagens. Sua estrutura é composta por um ORB com suporte a utilização de agentes móveis.

Voyager aliou Java à utilização de ORBs, de forma a possibilitar a integração de sistemas legados com a tecnologia de agentes móveis.

A proposta do *Voyager* é adicionar os recursos disponíveis nos sistemas baseados em agentes móveis e Java àqueles sistemas que já estavam em operação anteriormente, preservando assim os investimentos já feitos.

3.3.1 Elementos Básicos

Em sua versão 4.5, *Voyager* inclui os seguintes produtos:

- ***Voyager* ORB** é um ORB de alta performance que simultaneamente suporta a comunicação entre objetos *Voyager*, SOAP, CORBA, RMI e DCOM (*Distributed Component Object Model*).

- ***Voyager ORB Professional*** foi inteiramente construído nos moldes do *Voyager* ORB possuindo adicionalmente características como gerenciamento do console gráfico, configuração de framework, integração JNDI (*Java Naming and Directory Interface*), balanceamento de carga, diretórios replicados, serviços de nome CORBA e cliente altamente leve, possuindo apenas 15K.

- ***Voyager Security*** inclui um sistema flexível de segurança para a autenticação e autorização, suporte a comunicação em rede segura via SSL (*Secure Socket Layers*) e tunelamento.

- ***Voyager Transactions*** dá total suporte a elementos compatíveis ao OTS (*Object Transaction Service*) e JTA (*Java Transaction API*).

- ***Voyager Application Server*** é um poderoso servidor de ambiente baseado em EJB (*Enterprise JavaBean*) que separa a lógica de aplicação da de programação e suporte. *Voyager Application Server* também propicia uma ferramenta de serviços web que suporta JSP (*Java Server Pages*). As funcionalidades do *Voyager ORB Professional*, *Voyager Security* e *Voyager Transactions* são acionadas pelo *Voyager Application Server*.

3.3.2 Operações Fundamentais

Dentre as operações executadas em ambiente *Voyager* pode-se citar:

- **Migração**: *Voyager* tem habilidade de mover qualquer objeto Java presente em uma plataforma para outra. Entretanto somente objetos instanciados da classe agente podem, após mover-se, continuar sua execução no destino.

- **Persistência**: É definida uma interface *IVoyagerDb* que pode armazenar um objeto em um banco de dados relacional ou orientado a objetos.

- **Segurança**: Impõe, através de métodos, restrições a objetos originários de outras redes tais como impedir o monitoramento de uma determinada porta ou a criação de uma janela.

- **Ciclo de Vida**: O término da vida de um agente pode ser associado a um determinado fato ou tempo.

- **Mensagens e Eventos**: Podem ser enviadas mensagens síncronas, *one-way*, ou de uso futuro(*Future Messages*). Estas mensagens podem ser construídas tanto em tempo de execução quanto de compilação.

3.3.3 Segurança

Voyager tem sua segurança implementada pelo módulo *Security*, que impõe restrições aos objetos ou agentes, além daquelas já implementadas pela linguagem Java. Possui como mecanismos de segurança a utilização de *Socks* 4 e 5 e tunelamento HTTP. Não são incluídas no *Voyager* mecanismos de criptografia de *hosts* ou agentes.

4 Questões de segurança

4.1 Política de Segurança

Segundo [SOARES 98], uma política de segurança é um conjunto de leis, regras e práticas que regulam como uma organização gerencia, protege e distribui suas informações e recursos.

Um dado sistema é considerado seguro em relação a uma política de segurança, caso garanta o cumprimento das leis, regras e práticas definidas nessa política.

Uma política de segurança deve incluir regras detalhadas definindo como as informações e recursos da organização devem ser manipulados ao longo de seu ciclo de vida, ou seja, desde o momento que passam a existir no contexto da organização até que deixem de existir.

A implementação de uma política de segurança baseia-se na aplicação de regras que limitam o acesso de uma entidade às informações e recursos tendo como base a comparação de seu nível de autorização relativo a essa informação ou recurso e a designação da sensibilidade da informação. Assim, a política de segurança define o que é e o que não é permitido em termos de segurança, durante a operação de um dado sistema.

A base da política de segurança é a definição do comportamento autorizado para os indivíduos que interagem com o sistema.

O conjunto de regras que define uma política podem ser de dois tipos que analisam a natureza da autorização envolvida:

- regras baseadas em atributos de sensibilidade genéricos: nesta caso pode-se categorizar o recurso analisado. Os dados ou recursos são marcados com rótulos de segurança que indicam seu nível de sensibilidade, podendo ser classificados como confidencial, secreto ou ultra-secreto, por exemplo.

– regras baseadas em atributos individuais específicos: associa-se permissões de acordo com o nome ou o identificador de uma entidade do sistema.

Uma política definida por regras do primeiro tipo é denominado **Política de Segurança Baseada em Regras** já uma política definida com regras do segundo tipo é denominada **Política de Segurança Baseada em Identidade**. Cabe salientar que, dentro de uma mesma política, podemos ter um componente baseado em regras e outro baseada em identidade, ou seja, os dois tipos de políticas podem ser usados de forma complementar.

A autorização de uma política de segurança baseada em regras normalmente apóia-se em informações sobre sensibilidade. Em um sistema seguro, os dados ou recursos devem ser marcados com rótulos de segurança que indicam seu nível de sensibilidade. Os processos atuando sob o controle de indivíduos devem adquirir os rótulos de segurança apropriados, que definem o nível de autorização do indivíduo que o está controlando. As regras desse tipo de política utilizam os rótulos dos recursos e dos processos para determinar o tipo de acesso que pode ser efetuado. No caso de uma rede de computadores, os dispositivos que implementam os canais de comunicação também possuem rótulos de segurança. Nesse caso, as regras que definem a política de segurança também determinam quando é, ou não, permitido dados nesses canais, isto é, informações sensíveis só podem ser transmitidas em canais que ofereçam o nível de segurança adequado.

As políticas de segurança baseadas em identidade representam o tipo de controle de acesso mais encontrado nos computadores atuais. A base desse tipo de segurança é que um indivíduo ou processo, operando sob seu controle, pode especificar explicitamente os tipos de acessos que outros indivíduos podem ter às informações e recursos sob o seu controle. O objetivo desse tipo de política é permitir a implementação de um esquema de controle de acesso que possibilite especificar o que cada indivíduo pode ler, modificar ou usar. A idéia é implementar um mecanismo que permita descriminar, dentre os indivíduos que possuem nível de autorização suficiente para ter acesso a um recurso, aqueles que realmente necessitem realizar o acesso. Existem essencialmente duas formas de implementar esse tipo de política, dependendo de onde as informações sobre os direitos de acesso é armazenada: na entidade que está executando o acesso, ou como parte dos dados que estão sendo acessados. O primeiro caso é exemplificado com a utilização de "*capabilities*" possuídas pelos usuários e

utilizadas pelos processos atuando sobre seu controle. Listas de controle de acesso são exemplos desse último caso.

4.2 Arquitetura de Segurança

Segundo [RODRIGUES at al 02], o termo arquitetura de segurança de uma rede pode ser empregado com conotações diferentes. Para a ISO, uma arquitetura de segurança consiste na definição de conceitos e terminologia que formam um esquema básico para o desenvolvimento de protocolos. No caso da Internet, espera-se que a arquitetura de segurança forneça um conjunto de orientações voltado aos projetistas de redes e sistemas, sugerindo assim que esta arquitetura englobe não apenas definições de conceitos, como faz o padrão ISO, mas inclua adicionalmente orientações mais específicas sobre como e onde implementar os serviços de segurança na pilha de protocolos da Internet.

4.3 Modelo de Segurança

Um modelo de segurança representa o conjunto de todas as políticas de segurança que são adotadas para fazer com que as aplicações desenvolvidas e serviços executados hajam de forma segura. Java, por exemplo, adota o modelo de segurança conhecido como caixa de areia. A idéia deste modelo baseia-se no fato que pode-se fornecer um ambiente para o programa atuar, contudo restringe-se esta área de atuação dentro de limites pré-definidos de acordo com as características do usuário.

4.4 Tipos de Ataques

4.4.1 Ataques passivos

– **Análise de tráfego**: Este tipo de ataque normalmente usa um programa denominado monitor de comunicação. Este programa monitora as informações trocadas entre o sistema agente de agentes através da captura de mensagens e dos próprios agentes. Ela permite ao elemento que ataca o sistema gerar modelos de comunicação para uma análise posterior.

4.4.2 Ataques Ativos

- **Acesso Ilegal**: Neste ataque um agente acessa informações não permitidas. Isto pode ocorrer se o agente se fizer passar por outro usuário confiável do sistema ou se ele for executado em um ambiente não seguro. Neste último caso, por exemplo, um agente escrito em linguagem C pode usar um ponteiro aritmético para acessar locais arbitrários da memória.

- **Personificação**: Uma entidade se faz passar por outra. Em um exemplo típico, um agente penetra em um servidor usando a identidade de uma entidade confiável, obtendo desta forma, acesso a informações comerciais.

- **Cavalo de Tróia**: O cavalo de tróia é um agente executado por um usuário legítimo mas a ação executada por ele é diferente daquela que o usuário esperava.

- **Alteração**: As mensagens trocadas pelos agentes são alteradas e até mesmo excluídas durante seu trânsito pela rede. Informações modificadas de forma inesperada podem retornar falsos resultados.

- **Reenvio**: Uma cópia capturada de uma mensagem legítima enviada anteriormente é retransmitida para propósitos ilegítimos. Desta forma, a entidade mal-intencionada pode estar apta a obter resultados idênticos aqueles obtidos pelo agente original.

- **Exaustão de Recursos**: Um recurso é deliberadamente usado de forma tão intensa que a execução do servidor é interrompida. Este ataque também é conhecido como *Denial of Service* (DOS). O recurso em questão pode ser a banda da rede, assim como a memória ou a CPU.

- **Repudiação**: Uma entidade com a qual o sistema tinha se comunicado anteriormente, nega tê-lo feito.

4.5 Mecanismos de Segurança

Uma política de segurança pode ser implementada com a utilização de vários mecanismos. Vejamos agora, alguns daqueles mais utilizados:

4.5.1 Criptografia

A criptografia surgiu da necessidade de se enviar informações sensíveis através de meios de comunicações não confiáveis, ou seja, em meios onde não é possível garantir que o intruso não interceptará o fluxo de dados para a leitura (intruso passivo) ou para modificá-lo (intruso ativo). A forma de contornar esse problema é utilizar um método que modifique o texto original da mensagem a ser transmitida(texto normal), gerando texto criptografado na origem, através de um processo de codificação definido por um método de criptografia. O texto criptografado é então transmitido e, no destino, o processo inverso ocorre, isto é, o método de criptografia é agora aplicado para decodificar o texto criptografado transformando-o no texto original.

Dessa forma, sempre que um intruso conseguisse descobrir o método utilizado para encriptar os dados seria necessário substituir o método de criptografia. Essa substituição envolveria o desenvolvimento e a instalação dos procedimentos que implementam o novo método e treinamento de pessoal envolvido, tornando necessária a geração de um novo modelo.

Verificados estes problemas criou-se um novo modelo. Nele, o texto criptografado gerado a partir do texto normal , varia de acordo com a chave de codificação utilizada. Isto é, para um mesmo texto normal e um mesmo método de criptografia, chaves diferentes geram textos criptografados diferentes. Assim, o fato de um intruso conhecer o método de criptografia não é condição suficiente para que ele possa recuperar o texto original a partir do texto criptografado, pois para recuperar o texto original é necessário fornecer ao procedimento responsável pela decodificação , tanto o texto criptografado, quanto a chave de decodificação.

Um bom método de criptografia deve garantir que seja, senão impossível, pelo menos muito difícil que um intruso recupere, a partir do texto criptografado e do conhecimento sobre o método de criptografia, o valor das chaves. Sendo isso verdade, a confidencialidade do texto transmitido é garantida enquanto as chaves se mantiverem secretas.

Existem dois tipos de criptografia:

– **Criptografia com Chave Secreta**

Existem vários métodos de criptografia de chave secreta. O primeiro deles, atribuído a Júlio César, consiste em substituir a letra de uma mensagem pela terceira após a sua posição no alfabeto(considerando que o caracter que representa z seja a). Assim a seguinte mensagem: "Te encontro no cinema" seria apresentada da seguinte forma "Vh hqfrqvur qr fkqhpd".

Os métodos de criptografia que utilizam a mesma chave para a codificação e decodificação são chamados de simétricos ou baseados em chave secreta. O mais conhecido método simétrico existente é o DES(*Data Encriyptation Standart*). O principal problema do método DES, e de todos os algoritmos de criptografia simétricos é a exigência que o transmissor e o receptor de uma mensagem conheçam a chave secreta única usada na codificação e decodificação. O acordo entre a chave secreta entre o transmissor e o receptor, quando eles estão distantes um do outro, não é um problema trivial pois envolve a transmissão da chave e nem sempre é possível garantir que esta transmissão não seja interceptada. Se for interceptada, o responsável pelo ataque poderá ler todas as mensagens que serão criptografadas utilizando a referida chave "secreta".

– **Criptografia com Chave Pública**

Em 1976 Diffie e Hellman propuseram um novo método que revolucionou os sistemas de criptografia. O método baseia-se na utilização de chaves distintas: uma para a codificação e outra para a decodificação.

Uma vez respeitada essa condição, não há razão para não tornar a chave de codificação pública, simplificando bastante a tarefa do gerenciamento das chaves. Os métodos de criptografia que exibem essa característica são denominados assimétricos, ou baseados em chave pública.

O mais importante método de criptografia assimétrico é o RSA. O RSA baseia-se na dificuldade de se fatorar números primos muito grandes.

4.5.2 Assinatura Digital

O mecanismo de assinatura digital envolve dois procedimentos: assinatura de uma unidade de dados e verificação da assinatura em uma unidade de dados. O primeiro procedimento baseia-se em informação privada (única e secreta) do signatário. O segundo utiliza informação pública para reconhecer a assinatura. A característica essencial do mecanismo de assinatura digital é que ele deve garantir que uma mensagem assinada só possa ter sido gerada com informações privadas do signatário, portanto uma vez verificada a assinatura com a chave publica, é possível posteriormente provar a um terceiro que somente o proprietário da chave privada poderia ter gerado a mensagem.

4.5.3 Autenticação

A escolha do mecanismo de autenticação apropriado depende do ambiente onde se dará a autenticação.

Em uma primeira situação, os parceiros e os meios de comunicação são todos de confiança. Nesse caso a identificação de uma entidade pode ser confirmada por uma senha. Cabe mencionar que as senhas não protegem contra uso mal intencionado de ataques do tipo *replay*. Autenticação mútua pode ser implementada com a utilização de uma senha distinta em cada direção da comunicação.

Na segunda situação, cada entidade confia em seu parceiro, contudo não confia no meio de comunicação, deve-se usar criptografia na transmissão dos dados.

Na terceira situação, as entidades não confiam em seus parceiros(ou sentem que não é possível fazê-lo futuro) nem no meio de comunicação, deve-se usar criptografia na transmissão dos dados associada a assinatura digital.

4.5.4 Controle de Acesso

Os mecanismos de controle de acesso são utilizados para garantir que o acesso a um recurso é limitado aos usuários devidamente autorizados. As técnicas utilizadas incluem a utilização de listas de controle de acesso, que associam recursos a usuários autorizados, senhas e permissões associados aos recursos, cuja posse determina os direitos de acesso do usuário que as possui.

4.5.5 Integridade de Dados

Para garantir a integridade dos dados, podem ser usadas as técnicas de detecção de modificações, normalmente associadas com a detecção de erros em bits, em blocos, ou erro de seqüência introduzidos por enlace e redes de comunicação. Entretanto se os cabeçalhos e fechos carregando as informações de controle não forem protegidos contra modificações, um intruso que conheça as técnicas, pode contornar a verificação. Portanto para garantir a integridade é necessário manter confidenciais e íntegras as informações de controle usadas na detecção de modificações. Para controlar modificações na seqüência de unidade de dados transmitidas em uma conexão, deve-se usar técnicas que garantam a integridade das unidades de dados (garantindo que as informações de controle não sejam corrompidas) em conjunto com informações de controle de seqüência. Esses cuidados, embora não evitem a modificação da cadeia de unidades de dados garantem a detecção e notificação de ataques.

4.5.6 Preenchimento de Tráfego (*Traffic Padding*)

A geração de tráfego espúrico e o preenchimento das unidades de dados fazendo com que elas apresentem um comprimento constante são formas de fornecer proteção contra a análise do tráfego. Cabe ressaltar que o mecanismo de preenchimento de tráfego somente tem sentido caso as unidades de dados (ou pelo menos os campos de controle) sejam criptografados, impedindo que o tráfego espúrico, seja diferenciado do tráfego real.

4.5.7 Controle de Roteamento

A possibilidade de controlar o roteamento, especificando rotas preferenciais ou obrigatórias para a transferência de dados, pode ser utilizada para garantir que os dados sejam transmitidos em rotas fisicamente seguras ou para garantir que informação sensível seja transportada em rotas cujos canais de comunicação forneçam níveis

apropriados de proteção.

4.5.8 Rótulos de Segurança

Os recursos do sistema devem ser associados a rótulos de segurança que indicam, por exemplo, seu nível se sensibilidade. O rótulo de segurança deve ser mantido junto aos dados quando eles são transportados. Um rótulo de segurança pode ser implementado com adição de um campo aos dados.

5 Um modelo de segurança para agentes móveis

Este capítulo mostra o modelo de segurança desenvolvido por KARJOTH [KARJOTH at al 97], cujo objetivo é prover um *framework* para segurança de agentes móveis. O modelo suporta a definição de várias políticas de segurança e descreve como e onde um sistema seguro deve fazer uso destas políticas.

Inicialmente são discutidos os problemas que surgem quando se utiliza agentes móveis em aplicações. Posteriormente as soluções propostas pelo modelo são apresentadas. E, finalmente, baseado neste modelo de segurança, o capítulo 6 apresenta os requisitos que precisam ser analisados durante o projeto e desenvolvimento de uma aplicação que faça uso de agentes móveis para que a mesma possa ser considerada segura.

Com o intuito de facilitar o entendimento, a política definida encontra-se de acordo com as entidades associadas a aplicação desenvolvida no próximo capítulo. Brevemente pode-se dizer que a aplicação possui o objetivo de obter dados de freqüência de funcionários de suas filiais para poder executar e distribuir sua folha de pagamento.

5.1 Terminologia características de entidades

A seguir são apresentados termos comuns à definição de um modelo de segurança e que são de suma importância para a compreensão das informações aqui apresentadas:

- Domínio de Segurança

São computadores que compartilham uma mesma política de segurança e o uso dos mesmos mecanismos de prevenção. As máquinas da Microsoft, por exemplo, possuem o domínio de segurança do NT.

– Entidades

Uma entidade pode ser uma pessoa, máquina ou organização. O código executado por uma entidade pode ser explícito ou implícito. Quando um usuário usa um navegador para acessar a Internet, por exemplo, o navegador executa as ações pelo usuário.

– *Principal*

Um *principal* é um objeto de aplicação ou estrutura que representa uma entidade. Um *principal* identifica a entidade que ele representa, como um certificado de chave pública ou um nome do usuário.

– Identidade

Identidades são informações que descrevem uma entidade. Uma identidade é representada por um *principal.* Geralmente, um certificado de chave pública possui as informações necessárias para o estabelecimento da identidade de uma entidade.

– Contexto de Segurança

Um contexto é a associação entre a *thread* de execução e a entidade(*principal*).

– Signatário

É um tipo particular de *principal* usado para gerar assinaturas digitais.

– Credencial

Uma credencial é a informação que estabelece a identidade de uma entidade, geralmente durante a autenticação.

– Preferências de segurança

Representam a política de segurança definida pelo proprietário do agente. Entretanto, como o agente precisa confiar no contexto para transportá-lo, as preferências de segurança não são mais do que declarações de intenções.

– Base de dados de políticas

  Representa as políticas definidas pelo Mestre de Contexto, ou seja a entidade que gerencia um determinado contexto.

– Arquitetura de Segurança

  Implementação do modelo de segurança pelo provimento de um conjunto de componentes e suas interfaces. Os dois componentes da arquitetura de segurança de agentes são: as preferências de segurança do proprietário do agente e a base de dados de políticas do Mestre do Contexto.

5.2 O Problema que o modelo busca resolver

Agentes móveis oferecem um novo paradigma para a computação distribuída, mas é necessário que sejam considerados além de seus benefícios, as ameaças à segurança que eles podem gerar. Estas ameaças são causadas tanto por agentes quanto por *hosts* mal-intencionados. Não havendo mecanismos capazes de prevenir ataques, um *host* poderia modificar o estado de um agente ou alterar informações que o mesmo carrega consigo.

Usualmente, agentes são utilizados na buscar informações em empresas concorrentes, dessa forma, a aplicação vai estar em contextos de diferentes organizações no qual não há confiança mútua. As operações dos sistemas de agentes móveis vão requisitar serviços de segurança aceitos por todas as partes envolvidas no processo. Estes serviços não poderão ser violados nem de forma acidental, tampouco de forma intencional.

Há quatro questões de segurança que envolvem os sistemas de agentes móveis. São elas:

– Proteção de *hosts*

- Um agente que está visitando o *host* executa um ataque no sistema buscando acessar e corromper arquivos existentes neste local ou interromper o seu funcionamento. Este tipo de ataque inclui: acesso ilegal, personificação, cavalos de tróia, exaustão de recursos e repudiação.

- Uma terceira entidade mal intencionada busca atacar o *host* enviando um grande número de agentes de forma a prejudicar seu funcionamento. Este tipo de ataque inclui exaustão de recursos e reenvio.

– Proteção de agentes
  - Um agente interage com outro de forma a tentar extrair informações privadas e interromper a execução do mesmo. Este tipo de ataque inclui acesso ilegal.

  - Um agente em viagem é atacado por um *host* não confiável que pode tentar extrair ou alterar informações privadas deste agente. Este tipo de ataque inclui acesso e execução ilegais e adulteração.

  - Uma terceira entidade mal intencionada busca atacar agentes alterando as mensagens trocadas entre os mesmos ou interceptar a comunicação entre eles de forma a revelar o conteúdo do agente. Este tipo de ataque inclui alteração e personificação.

– Proteção do agente durante o tráfego na rede: Um agente pode iniciar a clonagem e envio de si próprio indefinidamente de forma a tentar sobrecarregar a rede. Este tipo de ataque inclui exaustão de recursos.

Segundo [KARJOTH at al 97] existem requisitos de segurança para agentes e *hosts* que não podem ser cumpridos. É impossível, por exemplo:

– Esconder qualquer coisa dentro de um agente sem o uso de criptografia;
– Comunicar-se de forma secreta com um grupo de agentes;
– Prevenir adulteração de agentes, a não ser que um hardware confiável esteja disponível na plataforma do agente;

– Conseguir distinguir entre um agente e seu clone.

Estas limitações acima apresentadas, implicam em o agente não poder carregar consigo informações secretas de seu proprietário, como números de cartões de crédito e chaves privadas, uma vez que estas informações poderão ser utilizadas por *hosts* não confiáveis.

Além disso é impossível para um agente verificar:

– a existência de um interpretador adulterado;
– se o interpretador irá executar o agente corretamente;
– se o *host* irá executar o agente integralmente;
– se o *host* irá transmitir o agente como lhe foi solicitado

Agentes móveis escritos em Java têm acesso a todos os arquivos de classes na máquina hospedeira, dessa forma eles precisam confiam no interpretador Java para a sua correta execução.

Agentes podem não estar ciente da política de segurança do seu computador hospedeiro, aquele que estabelece um contexto para o agente ser executado, e como essa política é implementada. Outros necessitam ter conhecimento da política utilizada pelo sistema para poder projetar suas ações. Além disso, é imprescindível que os agentes possam criar suas próprias definições de seguranças, para controlar o acesso a seus próprios dados, ou para fazer auditoria de suas próprias atividades relevantes.

5.3 O que esperar de um modelo de segurança?

Uma arquitetura de segurança deve preencher algumas funcionalidades chave no contexto da segurança que garantam as partes essenciais da política de segurança. Estas partes incluem:

– proteger a transferência e a comunicação dos agentes baseado na política de segurança;
– efetuar o necessário controle de acesso e a auditoria da execução dos agentes;
– prevenir que agentes ou grupos de agentes interfiram no funcionamento de outros ou proibir o acesso não autorizado ao estado destes agentes e

– prevenir que os agentes interfiram na definição de sua trajetória durante sua transferência.

Requisitos adicionais podem ser encontrados nesta arquitetura de segurança:

– permitir o uso de diferentes algoritmos de criptografia;
– armazenar as informações confidenciais encriptadas e
– ser compatível com padrões de segurança distribuídos como IETF, X/Open e OMG.

Para isso as políticas devem especificar é necessário que os projetistas envolvidos na criação do projetos especifiquem:

– as condições sob as quais os agentes móveis podem acessar objetos correntes;
– a necessidade da autenticação de usuários e outras entidades que acessam o sistema;
– quais ações estas entidades autenticadas podem realizar; e quem pode delegar seus direitos e
– a segurança das comunicações requerida entre agentes móveis e entre contextos.

Entidade/*Principal*	**Descrição**
Agente	Instanciação do próprio agente (a *thread* de execução do agente)
Desenvolvedor do Agente	O autor de um programa agente (uma pessoa, uma companhia ou outro)
Proprietário do Agente	Indivíduo que lançou o agente (uma pessoa) ou o aquele que tem responsabilidade legal com o comportamento do agente.
Contexto	O ambiente de execução em que o agente é executado, (um processo, *thread* ou outro programa)
Desenvolvedor do Contexto	Autor de um programa de contexto/produto (uma pessoa, uma empresa ou outro)
Mestre de Contexto	O proprietário, o administrador ou operador do contexto (uma pessoa, uma empresa ou outro)
Autoridade de Domínio	O proprietário, o administrador ou o operador do domínio (uma pessoa, uma empresa ou outro)

Tabela 1 – Descrição dos *Principals*

Todos os *principals* aqui introduzidos, podem definir políticas de segurança, em particular o proprietário do agente e o mestre de contexto. Assim, um sistema seguro de agentes pode implementar todas as políticas de segurança definidas. Por exemplo, embora o proprietário do agente possa ter especificado que o tempo máximo de permanência em um contexto é de 10 segundos, o mestre de contexto pode estabelecer um limite de 5 segundos, fator este que diminuirá o limite dado pelo proprietário.

A hierarquia das políticas de segurança definida por diferentes entidades é a seguinte:

Hierarquia						
Desenvolvedor do Agente	<	Proprietário do Agente	<	Mestre do Contexto	<	Autoridade de Domínio

Tabela 2 - Hierarquia das Políticas de Segurança

Tal hierarquia indica que a Autoridade de Domínio estabelece as políticas básicas sobre a execução de agentes dentro de um dado contexto, o qual pode então ser refinado, mas não sobreescrito, pelo Mestre de Contexto, Proprietário do Agente e o Desenvolvedor do Agente.

Para cada contexto, uma política de segurança pode descrever o mecanismo de comunicação com os demais contextos nos quais o agente irá trafegar. Por exemplo, mesmo que o agente não requisite nenhuma proteção de segurança para sua transferência ao contexto destino, a política de segurança do contexto destino pode reivindicar o uso de SSL(*Secure Socket Layer*) com a autenticação do cliente.

Os contextos precisam se proteger de agentes e agentes devem estar impedidos de interferir na execução de outros elementos. Dessa forma, o contexto do agente possui um monitor que somente dará acesso a um recurso se tal acesso estiver de acordo com a política de controle de acesso definida pelo mestre de contexto.

O mestre de contexto configura políticas de autorização para os agentes que chegam ao seu contexto. Em geral esta é a hierarquia de políticas de autorização:

- nível geral para uma entidade não autenticada;
- de acordo com a sua confiabilidade para uma entidade autenticada;
- de acordo com os agentes nos demais casos.

Adicionalmente, a autorização pode ser dada de acordo com o poder de computação do sistema, seu nível de ocupação, código de certificação ou o tipo do agente.

De acordo com a política de segurança definida usando esta hierarquia, o monitor do agente permite a leitura, execução e exclusão de arquivos locais, conexões com a rede, carregamento de bibliotecas ou a criação de janelas *pop-up*. Estes recursos devem estar definidos de acordo com o modelo Java e recursos adicionais são inseridos no contexto dos agentes, como por exemplo, a criação de um agente, seu envio a outro contexto, assim como a sua finalização.

As preferências de segurança dos agentes devem descrever quem e sobre quais circunstâncias um outro contexto de agente pode finalizar, clonar ou enviar um agente. As preferências podem ajudar a definir quais outros agentes podem chamar quais métodos.

Para aplicações críticas que associam credibilidade com uma versão específica do contexto, a identidade de um contexto deve ter um atributo, como um número serial de CPU. Exatamente como uma licença de software pode ser concedida a somente um computador específico, identificado por um número serial de CPU, um número serial de contexto se refere a uma permissão específica de seu software e hardware.

O mestre de contexto é também responsável por garantir que nenhum agente possa interferir em qualquer outro agente ativo no mesmo contexto, se não explicitamente permitido pelo proprietário do agente.

Como tanto o Mestre de Contexto quanto o Proprietário do Agente têm seus próprios interesses específicos concernentes a o que um agente deve ser capaz de fazer, ambos podem desejar restringir as capacidades um do outro. Tais restrições poderiam se aplicar a acessar os recursos locais de um contexto ou oferecer serviços a outros agentes. A base de dados e as preferências, portanto, constituem elementos poderosos para introduzir segurança dentro de um ambiente de agentes.

5.4 O modelo de segurança utilizado

Qualquer sistema de agentes móveis útil deve implementar políticas de segurança flexíveis e gerais. O modelo proposto por Karjoth [KARJOTH at al 97], simplifica a

administração dessas políticas por introduzir a noção de papéis, isto é, desenvolvedor, proprietário, mestre, autoridade e demais entidades.

O modelo de segurança identifica diversos *principals*, cada um tendo certas responsabilidades e interesses, que estão sumarizados na tabela 2.

- Existem três domínios: matriz, minhaFiliais e Inseguro
- Os seguintes contextos são criados: matriz, filialSJoseCampos, filialFpolis, filialCaxiasSul, filialTrezeTilhas, filialSBentoSul, filialSalvador, filialVitoria (todas pertencentes ao domínio minhasFiliais) e HostMalIntencionado (Inseguro).
- Dezesseis usuários têm acesso ao sistema: gerenteMatriz gerenteInformaticaFilialSJoseCampos, gerenteInformaticaFilialFpolis, gerenteInformaticaFilialCaxiasSul, gerenteInformaticaFilialTrezeTilhas, gerenteInformaticaFilialSBentoSul, gerenteInformaticaFilialSalvador, gerenteInformaticaFilialVitoria, desenvolvedorMatriz desenvolvedorFilialSJoseCampos, desenvolvedorFilialFpolis, desenvolvedorFilialCaxiasSul, desenvolvedorFilialTrezeTilhas, desenvolvedorFilialSBentoSul, desenvolvedorFilialSalvador e desenvolvedorFilialVitoria.

No que segue é descrita uma linguagem para definir políticas, usando os conceitos apresentados no modelo de segurança. Informações de como um Mestre de Contexto e um Proprietário de Agente podem usar esta linguagem para definir suas políticas serão apresentadas. A linguagem provê grupos nomeados, composição de entidades, ou seja, um conjunto de entidades e recursos hierárquicos com permissões associadas que possibilitam a definição de políticas de autorização de alto nível.

Para permitir controle de granularidade fina, uma política de segurança consiste de um conjunto de privilégios nomeados e um mapeamento de entidades - privilégios. Além disso, a linguagem permite a definição de listas que desabilitam agentes e contextos conhecidos como não confiáveis.

5.4.1 Linguagem de Autorização

Para ilustrar a linguagem estas entidades são definidas:

- **desenvolvedor:** desenvolvedorMatriz desenvolvedorFilialSJoseCampos, desenvolvedorFilialFpolis, desenvolvedorFilialCaxiasSul, desenvolvedorFilialTrezeTilhas, desenvolvedorFilialSBentoSul, desenvolvedorFilialSalvador e desenvolvedorFilialVitoria.

- **proprietário** – desenvolvedorMatriz desenvolvedorFilialSJoseCampos, desenvolvedorFilialFpolis, desenvolvedorFilialCaxiasSul, desenvolvedorFilialTrezeTilhas, desenvolvedorFilialSBentoSul, desenvolvedorFilialSalvador, desenvolvedorFilialVitoria e usuarioEspecialFilialFpolis

- **mestre** – gerenteMatriz, gerenteInformaticaFilialSJoseCampos, gerenteInformaticaFilialFpolis, gerenteInformaticaFilialCaxiasSul, gerenteInformaticaFilialTrezeTilhas, gerenteInformaticaFilialSBentoSul, gerenteInformaticaFilialSalvador e gerenteInformaticaFilialVitoria

- **autoridade** – gerenteMatriz

- **contexto**[1] – matriz, filial*

- **agente** –agente1, agente2

Por convenção, quando um agente é disparado, o contexto e o mestre se referem à entidade correspondente da máquina hospedeira local.

Aqui são usados nomes para simplificar a discussão, mas um valor real de **matriz** poderia ser algo como: **http://www.xyzSoftware.com** ou **http://150.172.31.23:4434**.

O nome do agente descrito como Agente1 poderia ser, de fato **http://150.172.31.23:4434. agentes.**frequenciaViaArquivo(filal, ano, mes) ou **http://150.172.31.23:4434. agentes.**frequenciaViaBanco(filal, ano, mes)

[1] O uso de caracteres curinga habilita a especificação de grupos de contexto, por exemplo, "**http://www.*.xyzSoftware.com**" .

As entidades descritas nesta linguagem podem ser de dois tipos: básicas e compostas.

Entidades Básicas

Entidades básicas endereçam agentes únicos ou grupos de agentes. Os seguintes são exemplos de entidades básicas:

- **agente = Agente2**, denota o agente chamado Agente2
- **proprietário = matriz**, denota todos os agentes disparados pela matriz.
- **desenvolvedor = desenvolvedorFilialSBentoSul**, denota os agentes escritos por desenvolvedores que trabalham na filial da empresa situada na cidade de São Bento do Sul.
- **contexto = filialCaxiasSul**, denota todos os agentes chegando diretamente da filial da empresa situada na cidade de Caxias do Sul.
- **mestre = gerenteMatriz**, denota todos os agentes chegando diretamente de contextos administrados pelo gerente da matriz da empresa.
- **Autoridade = gerenteMatriz**, denota todos os agentes chegando diretamente de contextos controlados pelo gerente da matriz da empresa.

Entidades Compostas

Entidades compostas oferecem um modo conveniente de combinar privilégios que devem ser garantidos à múltiplas entidades dentro de um único direito de acesso. Tal característica de agrupamento simplifica consideravelmente a administração de segurança. Membros em um grupo combinam várias entidades que devem ter os mesmos direitos de acesso.

Por exemplo, as seguintes especificações combinam entidades do mesmo tipo dentro de grupos nomeados e usam nomes de grupos em definições de regras, posteriormente:

GROUP Associação de Desenvolvedores =

desenvolvedorFilialSJoseCampos,

desenvolvedorFilialFpolis,

desenvolvedorFilialCaxiasSul,

desenvolvedorFilialTrezeTilhas,
desenvolvedorFilialSBentoSul,
desenvolvedorFilialSalvador
desenvolvedorFilialVitoria

Esta regra indica que o grupo "Associação de Desenvolvedores" consiste nos desenvolvedores presentes em todas as filiais da empresa.

desenvolvedorFilialSalvador IS_MEMBER_OF Associação de Desenvolvedores

Esta regra adiciona desenvolvedorFilialSalvador ao grupo Associação de Desenvolvedores.

Três outros construtores denotam diferença de conjuntos **(EXCEPT**), união de conjuntos (**OR**), e interseção de conjuntos (**AND**). A união de conjuntos é útil para manipular exceções, tais como um privilégio que deve ser dado a grupo, exceto para um certo usuário. O que segue são exemplos dessas construções:

proprietário = desenvolvedorFilialVitoria OR contexto = matriz

Representa qualquer agente de propriedade de desenvolvedorFilialVitoria ou chegando do contexto matriz.

proprietário = desenvolvedorFilialTrezeTilhas AND contexto = filialTrezeTilhas

Qualquer agente de propriedade de um desenvolvedor da filial situada na cidade de Treze Tilhas e chegando do contexto filialTrezeTilhas.

proprietario = Associação de Desenvolvedores
EXCEPT desenvolvedor = desenvolvedorFilialCaxiasSul

Qualquer agente disparado por qualquer membro de "Associação de Desenvolvedores" exceto aqueles escritos desenvolvedores da Filial da cidade de Caxias do Sul.

Privilégios

Privilégios definem as capacidades de executar código por se estabelecer restrições de acesso e limites sobre consumo de recursos. Um privilégio é um recurso (tal como um arquivo local), junto com permissões apropriadas tais como ler, escrever ou executar, no caso do arquivo local. A arquitetura de segurança aqui mostrada considera os seguintes tipos de recurso:

- **Arquivo** - arquivos no sistema de arquivo local
- **Rede** - acesso à rede
- **Janelas** - o sistemas de janelas local
- **Sistema** - qualquer tipo de recurso de sistema, tal como memória, e CPUs
- **QoP** - Qualidade de proteção
- **Contexto** - recursos do contexto
- **Agente** - recursos do agente.

Recursos são estruturados hierarquicamente. Assim, permissões podem ser dadas para um conjunto de recursos, ou mesmo para um tipo de recurso completo, como o acesso a um arquivo universal. Um exemplo com uma hierarquia simples é o tipo de recurso arquivo:

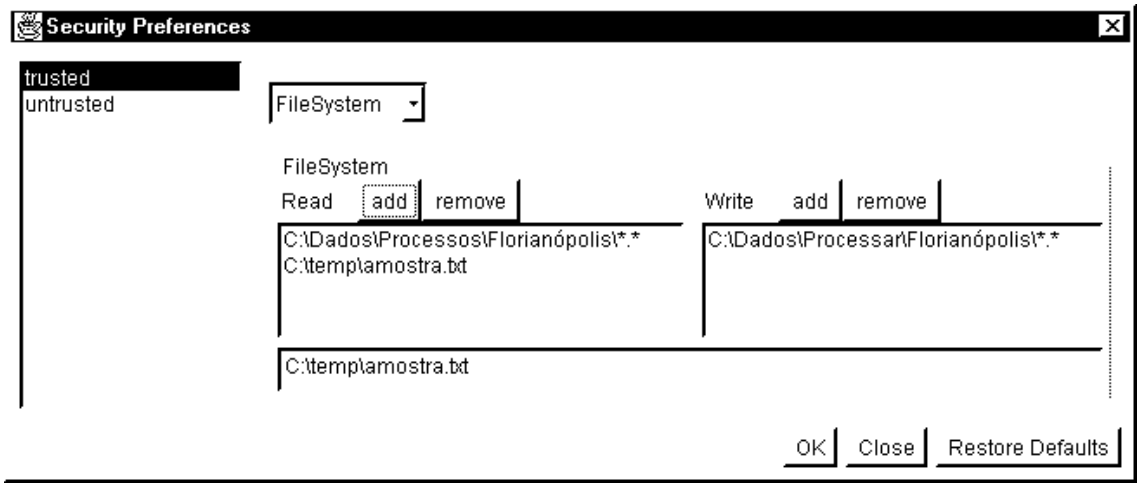

Figura 7 – Permitir acesso a arquivos

- **Arquivo – todos arquivos**
- **Arquivo – /temp/amostra.txt –** o arquivo **/temp/amostra.txt**

Acesso à rede é um recurso mais elaborado. A linguagem de autorização permite distinguir entre protocolos diferentes (por exemplo, TCP e HTTP) e selecionar portas ou faixas de portas para construir recursos:

- **Rede – qualquer tipo de rede**
- **Rede TCP – qualquer tipo de conexão TCP.**
- **Rede TCP host – conexões TCP para host.**
- **Rede TCP host porta – conexões TCP para host somente na porta.**

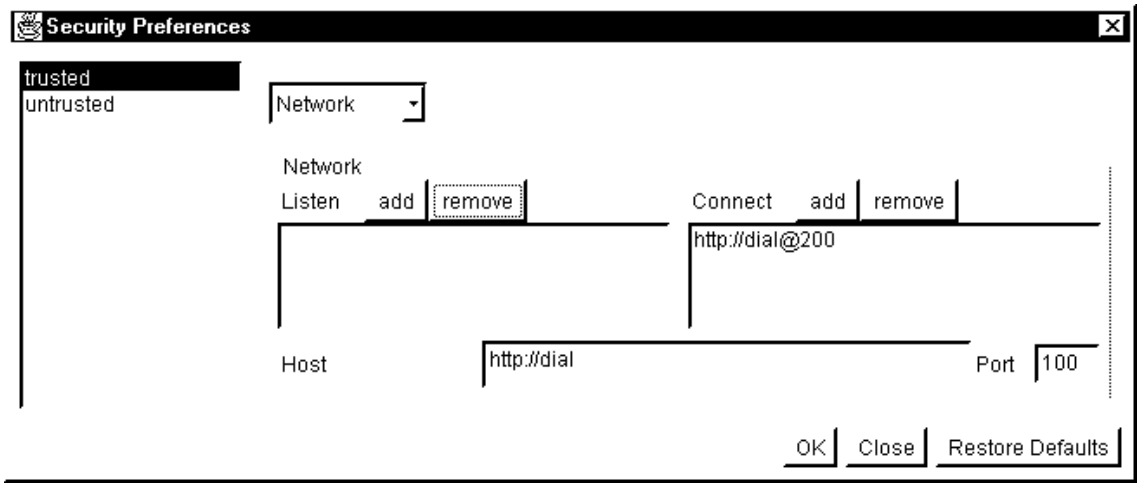

Figura 8 - Permitir conexões a rede

Cada recurso tem também um conjunto de permissões correspondentes. As permissões para redes são conectar e ouvir.

Os serviços providos pelo contexto do agente estão também sujeitos a controle. O contexto provê métodos para criar, retrair ou recolher e ativar agentes; enviar ou receber mensagens para/de outros agentes; para obter *proxies* de agente, a URL do *host*, *clips* de áudio e imagens; e obter ou estabelecer as propriedades do contexto.

Existem ainda permissões para acesso a banco de dados, apresentação de janelas e mensagens.

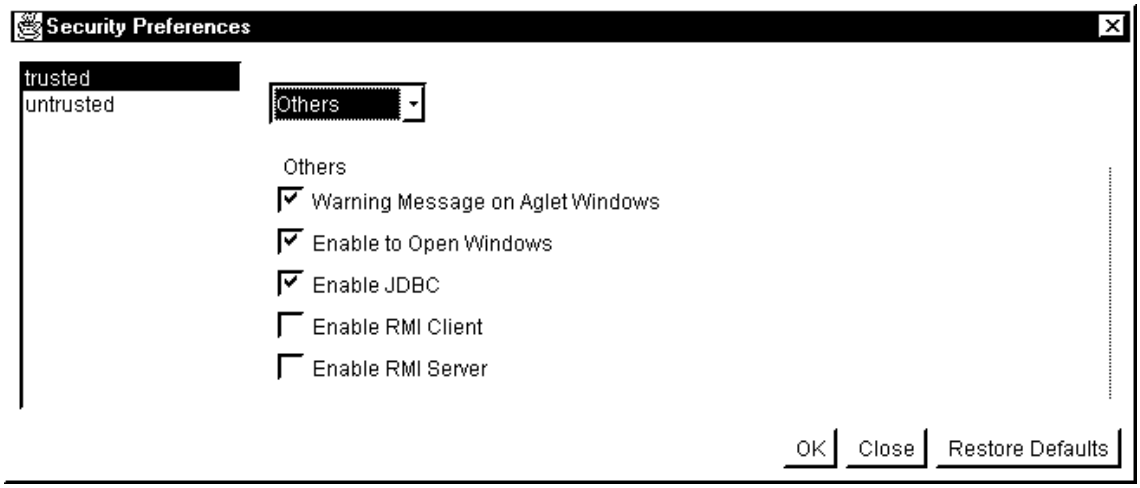

Figura 9 - Permitir conexões JDBC

Autenticação de Domínio

O servidor Tahiti está apto a verificar se um servidor externo pertence ao mesmo domínio de uma determinada máquina. Todos os servidores que pertencem ao mesmo domínio compartilham sua chave secreta e podem se autenticar mutuamente usando MAC (Message Authentication Code).

MAC funciona da seguinte forma: como os dois servidores possuem a mesma chave secreta. Quando o servidor externo envia uma mensagem declarando sua identificação, por exemplo, para o servidor local. Ele calcula o MAC em função da mensagem de identificação e da chave, onde MAC = Ck(M). A mensagem acrescida do MAC é transmitida ao servidor local .

O servidor local executa o mesmo cálculo com a mensagem, usando a mesma chave secreta, para gerar um novo MAC. O MAC resultante é comparado ao recebido anteriormente.

Se nenhuma entidade com interesses escusos souber da chave secreta destes domínios e se o MAC calculado for igual, o servidor local tem a certeza que o servidor externo é de seu próprio domínio

Para usar a autenticação de domínio, cada usuário ou administrador necessita obter a chave secreta da autoridade de domínio. A autoridade de domínio é responsável pela geração da chave para um servidor específico. A chave secreta compartilhada é associada com a senha do usuário. Então é solicitada ao usuário a informação de seu identificador de usuário e senha para torná-lo eficaz. O arquivo de chaves deve se manter secreto pois ele não é encriptado.

O maior problema no uso deste sistema é impossibilidade da distinção entre servidores válidos e servidores que roubaram chaves de outros.

Nos *Aglets*, a geração de chaves é um serviço oferecido pelo servidor Tahiti. Estas chaves podem ser importadas assim como exportadas para outro arquivo e, dessa forma as filiais da empresa poderão autenticar seu domínio.

Figura 10 - Criação de uma chave comum para os elementos do domínio inf.ufsc.br

Figura 11 - Arquivo contendo a chave do domínio

Esta política foi desenvolvida combinando recursos com permissões, os privilégios são definidos como segue:

Arquivo c:/temp/amostra.txt ler, escrever

um agente é permitido ler e escrever no arquivo **c:/temp/amostra.txt**.

Rede TCP matriz 200 conectar

o agente pode se conectar ao contexto matriz usando TCP na porta 200.

Sistema FREQUENCIA ufsc.db2.info

o agente pode carregar a biblioteca ufsc.db2.info.

Janelas Nível_Topo_janelas 1

o agente pode criar uma janela ao nível de topo.

A linguagem de autorização aqui usada introduz uma permissão. Contudo usa permissão pode ser usada como restrição pois trata-se de uma permissão negativa, excluindo certos agentes de executar no contexto.

Preferências do Mestre de Contexto

O mestre de contexto define a política de segurança para contextos de agentes sob seu controle. Esta política define as ações que um agente pode realizar. Na base de dados da política, o mestre de contexto combina entidades que denotam grupos de agentes e privilégios através de regras. A forma sintática de uma regra é:

<nome>:<entidade> → <privilégios>

Quando um agente corresponde a múltiplas entidades, dizemos que uma "regra de votação de consenso" combina as políticas para aquelas entidades. Em outras palavras, uma regra negativa rejeita a requisição. O conteúdo de uma base de dados de política de segurança poderia então ser semelhante a:

<Confiável>:
desenvolvedor = <Nome_Desenvolvedor> OR mestre = <Nome_Mestre_Contexto>
→

Arquivo <Nome_Arquivo> ler, escrever
Rede TCP <Nome_Contexto> aceitar
ApresentarJanelasSemBanner
Propriedades <recurso> ler
Sistema BIBLIOTECA <nome Biblioteca>
Sistema SEGURANÇA <obterPolítica>
Agente <Nome Agente> clonar, finalizar, despachar, recolher
Contexto Mensagem *

<Convidado>:

desenvolvedor = <Nome_Desenvolvedor> →

Rede <codebase> ouvir e resolver

Agente <*> clonar, finalizar, despachar e recolher

Contexto<Propriedades> ler

<NãoConfiável>:

desenvolvedor = <Nome_Desenvolvedor>, <Grupo_de_Entidades> →

Context NOT enter

Note que a nenhum dos agentes dentro do grupo NÃOCONFIÁVEL será permitida a entrada no contexto, porque eles não têm o privilégio necessário, ou seja, enter.

```
#Tahiti
#Thu Jun 20 01:37:39 GMT-02:00 2002
file.read=C:\\Dados\\Processos\\Florianopolis\\*.*;
          C:\\temp\\amostra.txt;
enable.rmiclient=false
socket.connect.http://dial@200=true
enable.jdbc=true
window.warning=true
file.write=C:\\Dados\\Processar\\Florianopolis\\*.*;
enable.rmiserver=false
window.open=true
```

Quadro 1 – Exemplo de política para um usuário confiável

```
#Tahiti
#Thu Jun 20 01:43:39 GMT-02:00 2002
enable.rmiclient=false
enable.jdbc=false
window.warning=true
enable.rmiserver=false
window.open=false
```

Quadro 2 – Exemplo de política para um usuário não confiável

Preferências do Proprietário do Agente

O proprietário do agente tem a oportunidade para estabelecer um conjunto de preferências de segurança que serão seguidas pelos contextos que um agente poderá visitar estas preferências estão contidas no próprio código do agente.

6 Criação de um modelo para o desenvolvimento de aplicações baseadas em agentes móveis compatível com o modelo de segurança

Para criar aplicações baseadas em agentes móveis que compreendam as características existentes no modelo de segurança apresentado anteriormente, é necessária a verificação de vários pontos durante o projeto e implementação da aplicação. Tais pontos fazem com que a aplicação se torne mais ou menos segura e como conseqüência o sistema como um todo torna-se proporcionalmente confiável.

Neste capítulo apresentamos as opções que o projetista da aplicação possui para criar seu sistema. Serão justificadas as razões pelas quais uma solução foi adotada em detrimento de outra.

6.1 O problema analisado

O sistema utilizado como exemplo para a criação do modelo é constituído de uma aplicação cujo objetivo é que a matriz de uma determinada empresa possa, usando agentes móveis, obter dados de freqüência de funcionários de suas filiais para que, de posse destes dados, a própria matriz possa gerar a folha de pagamento da empresa sem que os sistemas tenham que estar constantemente conectados.

Este exemplo usou uma empresa que possui filiais distribuídas em vários estados do pais. São sete filiais.

Um perspectiva para o uso da aplicação seria utilizá-la no Estado de Santa Catarina. Neste estado a folha de pagamento dos servidores é gerada pelo CIASC. Este

órgão tem a incumbência de obter dados dos funcionários e de acordo com suas características executar os cálculos necessários para gerar seus contra-cheques e disponibilizar valores nas contas de cada um dos funcionários. Neste caso, manter todos os órgãos constantemente conectadas é uma solução interessante, contudo onerosa. O consumo de recursos da rede para a busca de informações nem sempre é o adequado. Muitas vezes a solução indicada nestes casos é a utilização de VPNs.

6.2 A solução proposta

Esta aplicação busca uma solução prática, segura e estável para tal problema, para isso o primeiro passo adotado foi o desenvolvimento de uma aplicação que permitisse ao operador do sistema localizado nas filiais o cadastramento e consulta das informações dos funcionários existentes na filial.

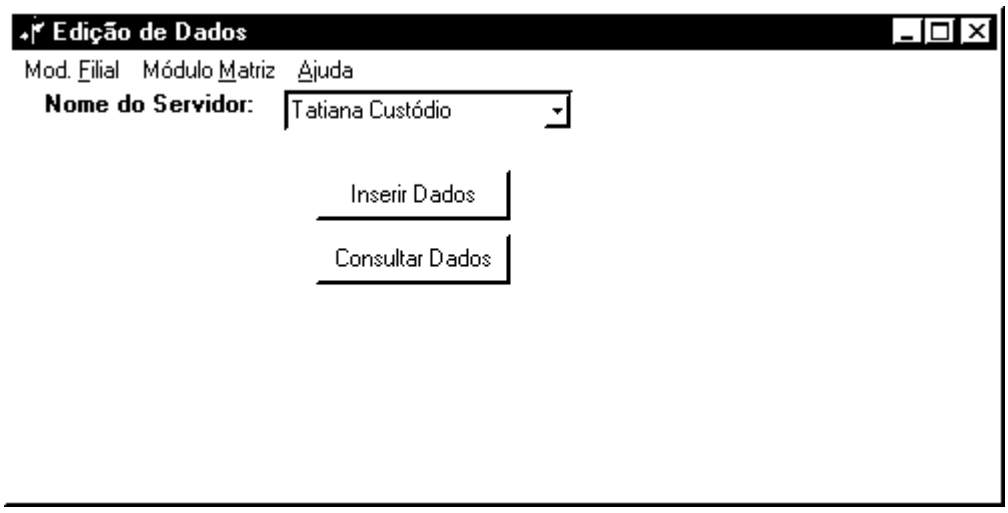

Figura 12 – Sistema de Controle de Freqüência – Filial

As informações referentes a freqüência do funcionário podem ser inseridas ou consultadas no sistema. As informações compreendem a quantidade de horas trabalhadas, o total de faltas, o total de atestados e as horas-extras trabalhadas pelo servidor.

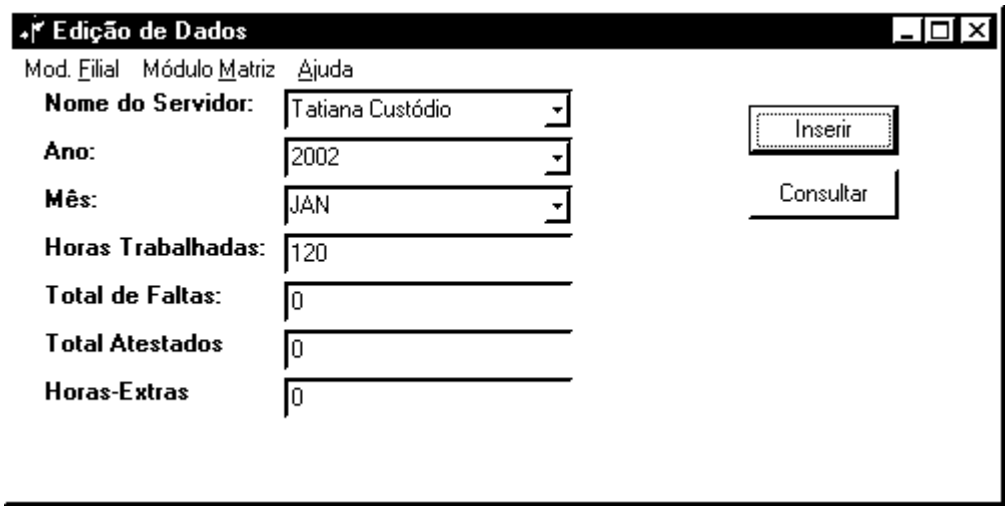

Figura 13 - Inserção de Dados de Freqüência

Os dados inseridos pelo operador são armazenados localmente no banco de dados da empresa. Este banco de dados está estruturado de acordo com a figura 14.

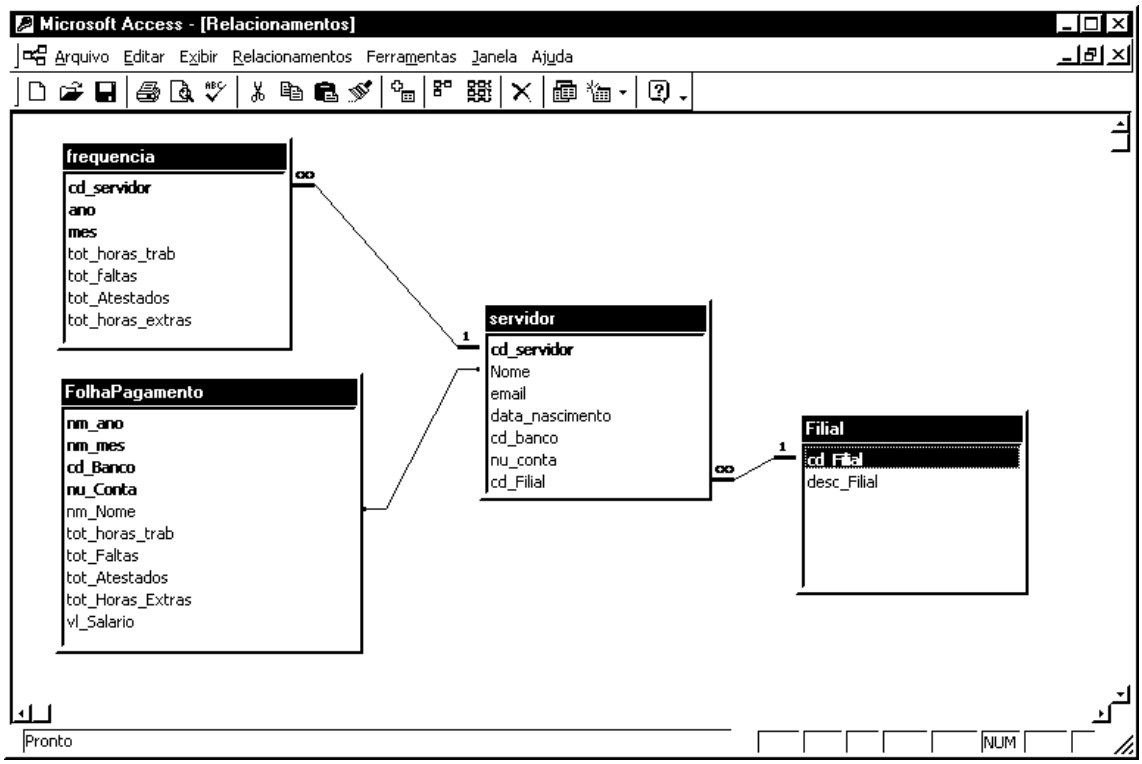

Figura 14 - Estrutura do banco de Dados

Após armazenados das informações no banco de dados da empresa os dados podem ser consultados de acordo com o funcionário selecionado pelo operador. Tal operação pode ser feita fazendo uso da seguinte interface:

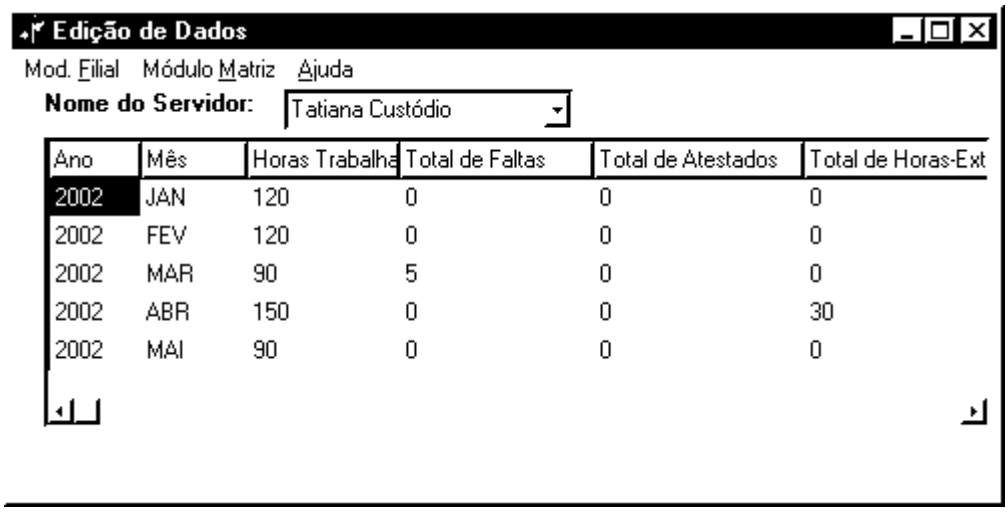

Figura 15 - Consulta de Informações no Sistema

Após os dados de um determinado mês terem sido gerados o próprio operador do sistema possui a opção de exportar dados. Esta opção executa as seguintes tarefas:

- Acessa o banco de dados para obter as informações de um determinado mês;
- Encripta as informações lidas no banco de dados com o mesmo sistema de encriptação/decriptação do software existente na matriz da empresa;
- Exporta os dados encriptados para um arquivo;
- Calcula o CRC das informações e insere no final do arquivo.

A utilização destes recursos permite que o agente que virá ao sistema para ler os dados leve consigo informações encriptadas sem ter a necessidade de levar chaves que permitam a leitura desta informação. A chave ou todo o algoritmo que permite a decriptação dos dados estará presente na aplicação existente na empresa matriz, como apresentado posteriormente.

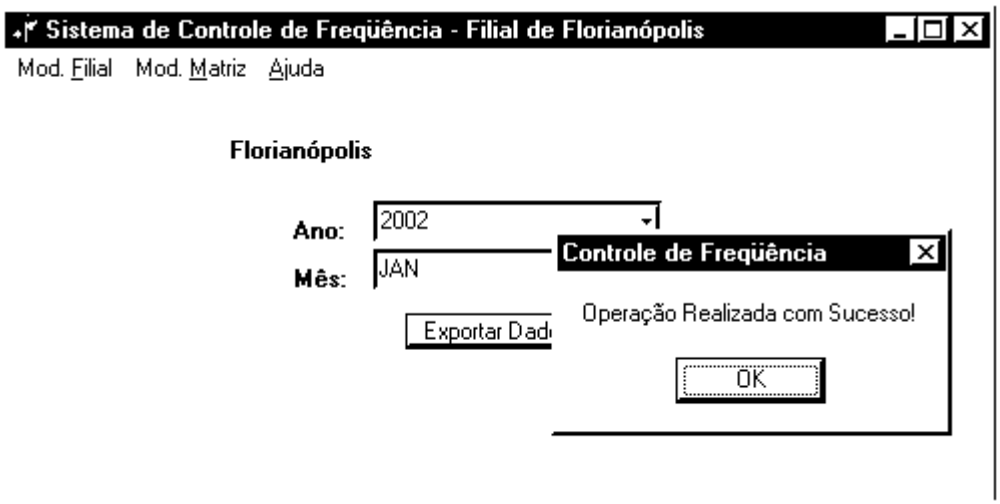

Figura 16 - Exportação de Dados para Arquivo

```
strSQL :=
'SELECT * FROM frequencia where ano = ' + quotedStr(cbAno.Text) + ' and mes = ' +
 quotedStr(cbMes.Text);

dadoCRC := '';
adoset:=ADOConnection1.Execute(strSQL);
If not adoset.EOF Then
 begin
  lista := TStringList.Create;
  while not adoset.EOF do
    begin
      valor := '';
      for J:= 0 to adoset.fields.Count-1 do
       begin
        adofield:=adoset.fields(J);
        valor:= valor + String(adofield.Value) + ';';
       end;
       valorEncriptado := Encriptar(valor);
       dadoCRC := dadoCRC + valorEncriptado
       lista.Add(valorEncriptado);
      adoset.movenext;
    end;
  crc := calcularCRC(dadoCRC);
  exportaParaArquivo(lista,cbAno.Text,cbMes.Text, crc);
  lista.Free;
  ShowMessage('Operação Realizada com Sucesso!');
 end
Else
 begin
   ShowMessage('Informações não localizadas. Impossível Exportar!');
 end;
```

Quadro 3 – Fragmento do código de exportação dos dados

Exportando estes dados o agente vindo da matriz da empresa saberá em que local da máquina filial serão encontrados os dados e assim poderá lê-los e leva-los consigo para a aplicação existente na empresa matriz que se encarregará de ler os dados decriptá-los, verificar se não houve erros durante a transmissão dos dados e incorpora-los ao sua tabela de folha de pagamento após executar os devidos cálculos.

Para que a agente vindo da matriz possa ter acesso aos dados é necessário que ele tenha a permissão de acesso a arquivos habilitada no menu de segurança do servidor *Tahiti*. Para o funcionamento da aplicação é essencial ainda que o servidor *Tahiti* esteja inicializado.

Contudo não há porque o servidor estar disponível durante todo o tempo se sabe-se que a busca de informações ocorre uma única vez a cada mês. Assim aplicação pode iniciar o servidor no dia definido pelo sistema.

```
 int posicao, totTrab, totFalta, totAtestados, totHoraExtra;
 String nome, banco, conta, strSQL, url;
 double salario;

String localAcesso = ".."
if (dadosRetorno.length()==0)
    {
      setText("Dados retonaram vazios da máquina remota");
      waitMessage(2 * 1000);
      dispose();
    }
    String txtArq = "";
    String arq= "localAcesso" ;
     try
     {
        FileOutputStream f1 = new FileOutputStream(arq);
        PrintStream o1 = new PrintStream(f1);
        o1.println(dadosRetorno);
    }
    catch(Exception e)
    {
     setText(String.valueOf(e));
    }
```

Quadro 4 – Obtenção do Arquivo pelo Agente

Uma vez o agente estando de posse destas informações, retorna a matriz e salva estes dados no formato de arquivo em um diretório previamente definido. Como pode-se ver no quadro a seguir a estrutura do agente tornou-se extremamente simples, cabendo a própria aplicação todo o processamento necessário ao funcionamento.

```
String txtArq = "";
       String arq= localEscolhido ;
       try
       {
           FileInputStream f1 = new FileInputStream(arq);
           DataInputStream i1 = new DataInputStream(f1);
           String val = "";
           while(val != "-1")
           {
             txtArq = txtArq + val;
             val = i1.readLine();
           }
           Message mensagem = new Message("Retorna");
           mensagem.setArg("dados",txtArq);
           handleMessage(mensagem);
       }
       catch(Exception e)
       {
         setText(String.valueOf(e));
       }
```

Quadro 5 – Agente escrevendo informações na matriz

A aplicação existente na matriz pode agora incorporar os dados recebidos através do agente em seu sistema de folha de pagamento.

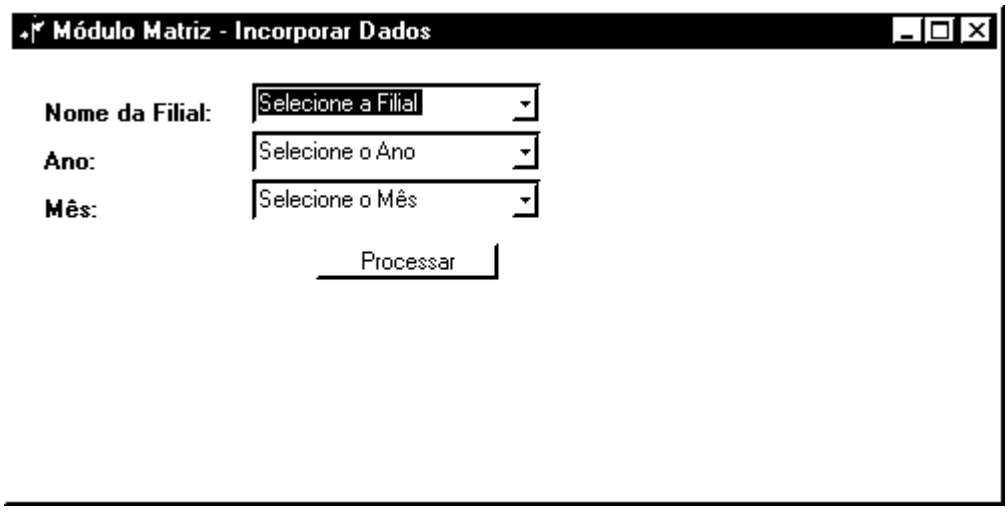

Figura 17 – Incorporar dados trazidos pelo agente no sistema

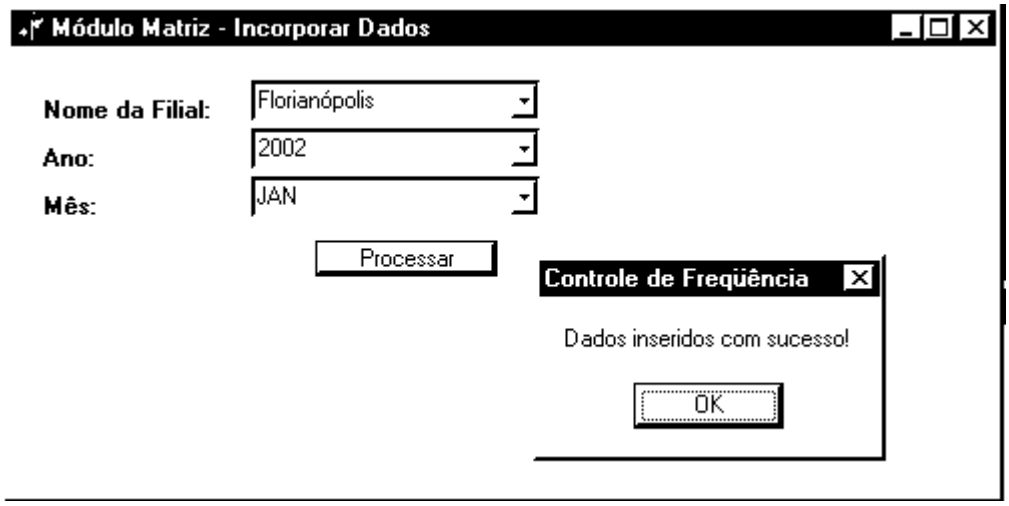

Figura 18 - Sucesso na incorporação dos dados

Neste ponto o CRC é calculado, os dados são decriptados, o valor do salário é definido e inserido no banco de dados, assim como uma cópia deste arquivo é guardada como histórico da operação, como apresentado no quadro abaixo.

```
    arquivo := 'C:\Dados\Processar\'+ cbFilial.Text +'\'+cbAno.Text+cbMes.Text +'.txt';
    AssignFile(arq, arquivo);
    Reset(arq);
    Inicio := true;
    CRCOK := false;
    while not EOF(arq) do
     begin
      readln(arq,dados);
      If inicio Then
       begin
        CRCOK := verificaCRC(dados);
        Inicio := false;
       End;
      If CRCOK Then
       begin
        dadoDecriptado := frmControleDeFrequencia.Decriptar(dados);
        InsercaoOK :=  FolhaDePagamento.inseriuItem(dadoDecriptado);
        If not InsercaoOK Then
          deuErro := true;
       end;
       CloseFile(arq);
      End
     Else
      DeuErro := true;
    If deuErro Then
      ShowMessage('Ocorreram erros durante o processo de inserção dos dados! Por favor
reprocesse o arquivo ou verifique se ele não foi corrompido durante a transmissão!')
    Else
      Present:= Now;
      DecodeDate(Present, Year, Month, Day);
      novoNomeArquivo := 'C:\Dados\Log\'+ cbFilial.Text + intToStr(Year) + intToStr(Month)
+cbAno.Text+cbMes.Text +'.txt';
       If CopyFile(PChar(arquivo),PChar(novoNomeArquivo), false) Then
        ShowMessage('Dados inseridos com sucesso!')
      Else
       ShowMessage('Dados inseridos com sucesso, contudo aquivo de Log não pode ser
corretamente criado!');
   end;
```

Esta mesma aplicação foi desenvolvida utilizando o agente para consultar diretamente o banco de dados, contudo esta solução não foi considerada adequada pois:

- a aplicação precisaria armazenar os dados de forma encriptada no banco de dados para que o agente não precisasse levar consigo o algoritmo de encriptação e as chaves de acesso aos dados;
- a senha de acesso ao banco de dados teve que ser enviada junto ao agente, e enquanto não houverem formas de conseguir encriptar o agente completamente esta solução torna-se inadequada.

Resumidamente o modelo de desenvolvimento de aplicações utilizando agentes móveis considera que o projetista do sistema deve levar em consideração os seguintes poontos:

- sempre adotar uma política de segurança que proteja seus dados. Definir elementos considerados confiáveis e não confiáveis para o sistema;
- verificar se a aplicação necessita estar disponível 24 horas por dia, se isso não ocorrer é indicado o uso de uma aplicativo para *Desktop* juntamente com o agente;
- evitar que a aplicação tenha que acessar diretamente o banco de dados da empresa. Se isso for realmente necessário, manter uma cópia atualizada e esta cópia deverá ser acessada pelos agentes;
- enviar dados encriptados;
- calcular CRC dos dados, e
- se for possível usar uma aplicação de auxílio ao desenvolvimento manter todo o processamento pesado nesta aplicação. Dessa forma o agente será simples e leve.

7 Conclusões

Como qualquer código executável através de *download*, agentes móveis são ameaças potenciais ao sistema. Contudo eles podem também ser ameaçados pelos *hosts* nos quais trafegam; situação incomum se comparada com os demais paradigmas da computação em rede.

Como a disseminação da utilização de agentes móveis em aplicações está intimamente relacionada à garantia da segurança na utilização do mesmo, um modelo de segurança eficaz é essencial para uma melhor aceitação dos agentes como solução para aplicações distribuídas.

O modelo de segurança apresentado por Karjoth [KARJOTH at al 97] é o primeiro passo na busca dessas garantias. O modelo claramente define as entidades associadas às suas responsabilidades e interesses, explicando como os agentes migram e executam acesso ao sistema local das máquinas visitadas. Dois elementos foram inseridos na arquitetura de segurança: Preferências de Segurança e a Base de Dados de Políticas e demonstrou-se como estes elementos controlam as questões de segurança sem que o agente tenha consciência do fato. O modelo procura proteger o *host* e os agentes.

A encriptação do agente como um todo é um problema de difícil solução no momento atual, dessa forma é necessário que se encontrem alternativas para que os dados sensíveis que precisam ser enviados juntamente com os agentes sejam protegidos. Estes dados compreendem senhas, cartões de crédito, chaves privadas, entre outros.

Este trabalho buscou uma alternativa para este problema ou seja, fazer uso de uma aplicação para "*desktop*" processasse as informações solicitadas pelo agente retirando de sua responsabilidade questões como encriptação de dados, proteção de informações

sensíveis, uma vez que ele não precisaria levar consigo tais dados. Dessa forma o agente tornou-se apenas um instrumento que leva a informação através da rede e não todo o processamento.

Sob vários aspectos características interessantes dos agentes deixaram de ser utilizadas como por exemplo o fato de ele levar consigo todo o "know-how" do processamento, contudo o ganho sob o aspecto segurança supera em muito a ausência destas características.

Como dito anteriormente, ainda não há uma solução para a proteção do estado interno de um agente contra. Assim, uma extensão deste trabalho, propõe-se a apresentar soluções para tópicos aqui não abordados como o acima descrito e questões como o consumo de recursos da máquina hospedeira, por exemplo.

8 Anexos

chaves geradas para a autenticação de domínios

```
Secret:
f76e9f4a26739aaab601db9fc19bc1f85458f8ef3505ba91e649380f54bd6e13
Date: 2001.08.13 03:52:31.699 GMT-03:00
Domain: Aglets Sample Domain
Owner: asdkprovider
Signature: 302c02146866abdafbb949aa05a3ab1e3ce0331ea20f7ffb0214
40927ff702f4e3c9b552a3bb
02e8ded8f955a1cd

Secret:
0b672416ced639d697c833f050d7766c4c27a59b55c05715f08a63c723496777
Date: 2002.03.24 20:18:52.960 GMT-03:00
Domain: *.inf.ufsc.br
Owner: matriz
Signature:
302c021411622de4a69e2a9710517a040dc7ce842e30db7f0214548d7e4c48
d1106af4975f857384eda71e29430f
```

arquivo.java

```
package examples.dissertacao;

import com.ibm.aglet.*;
import com.ibm.aglet.AgletProxy;
import com.ibm.aglet.event.*;
import com.ibm.aglet.util.*;


import java.lang.InterruptedException;
import java.io.*;
import java.net.*;
import java.awt.*;
public class Arquivo extends Aglet {


    transient Frame my_dialog = new MyDialog(this);
    String message = "Consultar";
    String home = null;
    boolean souRemoto = false;
    String anoEscolhido;
    String mesEscolhido;
    String filialEscolhida;
```

```
    String dadosRetorno;


    public void onCreation(Object init) {
        my_dialog.pack();
        my_dialog.setTitle("Obter Dados de Frequência");
        my_dialog.resize(my_dialog.preferredSize());
        my_dialog.show();

        // Initialize the variables.
        home = getAgletContext().getHostingURL().toString();
        addMobilityListener(
           new MobilityAdapter(){
            public void onDispatching(MobilityEvent e)
            {
              setText("Enviando Agente para o Host Remoto...");
              waitMessage(2 * 1000);
            }
            public void onArrival(MobilityEvent e)
            {
              souRemoto = true;
            }
           }
       );
    }


   public void run()
   {

     if (home.equals(getAgletContext().getHostingURL().toString()))
     {
       setText("Estou em casa...");
       waitMessage(2 * 1000);
       int posicao, totTrab, totFalta, totAtestados, totHoraExtra;
       String nome, banco, conta, strSQL, url;
       double salario;
       if (dadosRetorno.length()==0)
       {
         setText("Dados retonaram vazios da máquina remota");
         waitMessage(2 * 1000);
         dispose();
       }
        String txtArq = "";
        String arq= "C:\\Dados\\Processar\\" + filialEscolhida +"\\" +
anoEscolhido  + mesEscolhido + ".txt" ;
        try
        {
           FileOutputStream f1 = new FileOutputStream(arq);
           PrintStream o1 = new PrintStream(f1);
           o1.println(dadosRetorno);
       }
       catch(Exception e)
       {
        setText(String.valueOf(e));
       }
     }
     else
     {
       setText("Estou no host remoto...");
       String txtArq = "";
```

```
       String arq= "C:\\Dados\\Processos\\" + filialEscolhida +"\\" +
anoEscolhido  + mesEscolhido + ".txt" ;
       try
       {
           FileInputStream f1 = new FileInputStream(arq);
           DataInputStream i1 = new DataInputStream(f1);
           String val = "";
           while(val != "-1")
           {
             txtArq = txtArq + val;
             val = i1.readLine();
           }
           Message mensagem = new Message("Retorna");
           mensagem.setArg("dados",txtArq);
           handleMessage(mensagem);
       }
       catch(Exception e)
       {
         setText(String.valueOf(e));
       }
     }
   }


    public void Envia(Message msg)
    {
         String destino = String.valueOf(msg.getArg("destino"));
         String mes = String.valueOf(msg.getArg("mes"));
         mesEscolhido = mes;
         String ano = String.valueOf(msg.getArg("ano"));
         anoEscolhido = ano;
         String fil = String.valueOf(msg.getArg("filial"));
         filialEscolhida = fil;
      try
      {
         dispatch(new URL (destino));
      }
      catch(Exception e)
      {
         setText("Erro ao enviar o Agente ao Host Remoto");
      }
    }


    public void Recebe(Message msg)
    {
      dadosRetorno = String.valueOf(msg.getArg("dados"));
      try
      {
         dispatch(new URL (home));
      }
      catch(Exception e)
      {
         setText("Erro ao enviar o Agente de volta a Origem");
      }
    }


    public boolean handleMessage(Message msg) {
        if (msg.sameKind("Envia")) {
           Envia(msg);
```

```
        } else if (msg.sameKind("Retorna")) {
           Recebe(msg);
        } else {
            return false;
        }
        return true;
    }


       public void dialog(Message msg) {
        // check and create a dialog box
        if (my_dialog == null) {
            my_dialog = new MyDialog(this);
            my_dialog.pack();
            my_dialog.resize(my_dialog.preferredSize());
        }

        // show the dialog box
        my_dialog.show();
    }
}


class MyDialog extends Frame {

    private Arquivo aglet       = null;
    private Choice filial       = new Choice();
    private Choice ano          = new Choice();
    private Choice mes          = new Choice();
    private Label  diretoria    = new Label();
    private Label  anoL         = new Label();
    private Label  mesL         = new Label();
    private TextField msg       = new TextField(18);
    private Button go           = new Button("Enviar");
    private Button close        = new Button("Fechar");

    MyDialog(Arquivo aglet) {
        this.aglet = aglet;
        layoutComponents();
    }

    private void layoutComponents() {
        aglet.message ="";

        // Layouts components
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cns = new GridBagConstraints();
        setLayout(grid);

        cns.weightx = 0.5;
        cns.ipadx = cns.ipady = 5;
        cns.fill = GridBagConstraints.HORIZONTAL;
        cns.insets = new Insets(5,5,5,5);

        cns.weightx = 1.0;
        cns.gridwidth = GridBagConstraints.REMAINDER;
        grid.setConstraints(filial, cns);
        add(diretoria);
        add(filial);

        cns.gridwidth = GridBagConstraints.REMAINDER;
```

```
        cns.fill = GridBagConstraints.BOTH;
        cns.weightx = 1.0;
        cns.weighty = 1.0;
        cns.gridheight = 2;
        grid.setConstraints(ano, cns);
        add(anoL);
        add(ano);

        cns.gridwidth = GridBagConstraints.REMAINDER;
        cns.fill = GridBagConstraints.BOTH;
        cns.weightx = 1.5;
        cns.weighty = 1.5;
        cns.gridheight = 3;
        grid.setConstraints(mes, cns);
        add(mesL);
        add(mes);

        cns.weighty = 0.0;
        cns.fill = GridBagConstraints.NONE;
        cns.gridheight = 1;

        Panel p = new Panel();

        grid.setConstraints(p, cns);
        add(p);
        p.setLayout(new FlowLayout());
        p.add(go);
        p.add(close);
        diretoria.setText("Filial:");
        anoL.setText("Ano:");
        mesL.setText("Mês:");
        filial.addItem("São José dos Campos");
        filial.addItem("Florianópolis");
        filial.addItem("Caxias do Sul");
        filial.addItem("Treze Tilhas");
        filial.addItem("São Bento do Sul");
        filial.addItem("Salvador");
        filial.addItem("Vitória");
        ano.addItem("2002");
        ano.addItem("2001");
        ano.addItem("2000");
        mes.addItem("Jan");
        mes.addItem("Fev");
        mes.addItem("Mar");
        mes.addItem("Abr");
        mes.addItem("Mai");
        mes.addItem("Jun");
        mes.addItem("Jul");
        mes.addItem("Ago");
        mes.addItem("Set");
        mes.addItem("Out");
        mes.addItem("Nov");
        mes.addItem("Dez");
    }



    public boolean handleEvent(Event ev) {
        if (ev.id == Event.WINDOW_DESTROY) {
            hide();
```

```
            return true;
        }
        return super.handleEvent(ev);
    }

    public boolean action(Event ev, Object obj) {
        String destino = "";
        String aURL[] = new String[7] ;
        aURL[0] = "atp://dial:100";
        aURL[1] = "atp://dial:200";
        aURL[2] = "atp://dial:300";
        aURL[3] = "atp://dial:400";
        aURL[4] = "atp://dial:500";
        aURL[5] = "atp://dial:600";
        aURL[6] = "atp://dial:700";
        if (ev.target == go) {
            String filialConsultada = filial.getSelectedItem();
            int item = filial.getSelectedIndex();
            destino = aURL[item];
            String anoConsultado = ano.getSelectedItem();
            String mesConsultado = mes.getSelectedItem();
           Message mensagem = new Message("Envia");
            mensagem.setArg("destino",destino);
            mensagem.setArg("ano",anoConsultado);
            mensagem.setArg("mes",mesConsultado);
            mensagem.setArg("filial",filialConsultada);
            aglet.handleMessage(mensagem);
        } else if (ev.target == close) {
            hide();
        } else {
            return false;
        }
        return true;
    }
}
```

Banco.java

```
package examples.dissertacao;

import com.ibm.aglet.*;
import com.ibm.aglet.AgletProxy;
import com.ibm.aglet.event.*;
import com.ibm.aglet.util.*;


import java.lang.InterruptedException;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.sql.*;
public class Banco extends Aglet {


    transient Frame my_dialog = new MyDialog(this);
    String message = "Consultar";
    String home = null;
    boolean souRemoto = false;
    String anoEscolhido;
    String mesEscolhido;
    String dadosRetorno;
    private double valorHora = 40.65;
    private double valorHoraExtra = 78.22;

    public void onCreation(Object init) {
        my_dialog.pack();
        my_dialog.setTitle("Obter Dados de Frequência");
        my_dialog.resize(my_dialog.preferredSize());
        my_dialog.show();

        // Initialize the variables.
        home = getAgletContext().getHostingURL().toString();
        addMobilityListener(
           new MobilityAdapter(){
            public void onDispatching(MobilityEvent e)
            {
              setText("Enviando Agente para o Host Remoto...");
            }
            public void onArrival(MobilityEvent e)
            {
              souRemoto = true;
            }
           }
       );
    }


   public void run()
   {
     Connection con;
     Statement stmt;
     PreparedStatement pstmt;
     ResultSet rs;
     if (home.equals(getAgletContext().getHostingURL().toString()))
     {
       int posicao, totTrab, totFalta, totAtestados, totHoraExtra;
```

```
        String nome, banco, conta, strSQL, url;
        double salario;
        if (dadosRetorno.length()==0)
        {
          setText("Dados retonaram vazios da máquina remota");
          waitMessage(2 * 1000);
          dispose();
        }
        try
        {
         while (dadosRetorno.length() > 0 )
          {
          posicao = dadosRetorno.indexOf(";");
          nome = dadosRetorno.substring(0,posicao);
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          banco = dadosRetorno.substring(0,posicao);
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          conta = dadosRetorno.substring(0,posicao);
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          totTrab =
Integer.valueOf(dadosRetorno.substring(0,posicao)).intValue();
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          totFalta =
Integer.valueOf(dadosRetorno.substring(0,posicao)).intValue();
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          totAtestados =
Integer.valueOf(dadosRetorno.substring(0,posicao)).intValue();
          dadosRetorno = dadosRetorno.substring(posicao +
1,dadosRetorno.length());
          posicao = dadosRetorno.indexOf(";");
          totHoraExtra =
Integer.valueOf(dadosRetorno.substring(0,posicao)).intValue();
          if (posicao == dadosRetorno.length())
          {
            dadosRetorno = "";
          }
          salario = (((totTrab + (6*totAtestados) - (6*totFalta)) *
valorHora) + (totHoraExtra * valorHoraExtra));
          Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
          url   = "jdbc:odbc:" + "rh";
          con = DriverManager.getConnection(url, "", "");
          strSQL = "INSERT INTO FolhaPagamento
(nm_ano,nm_mes,cd_Banco,nu_Conta,nm_Nome,
tot_horas_trab,tot_Faltas,tot_Atestados,tot_Horas_Extras,vl_Salario)VALUES
(?,?,?,?,?,?,?,?,?,?)";
          pstmt = con.prepareStatement(strSQL);
          pstmt.setString(1,anoEscolhido);
          pstmt.setString(2,mesEscolhido);
          pstmt.setString(3,banco);
          pstmt.setString(4,conta);
          pstmt.setString(5,nome);
```

```
          pstmt.setInt(6,totTrab);
          pstmt.setInt(7,totFalta);
          pstmt.setInt(8,totAtestados);
          pstmt.setInt(9,totHoraExtra);
          pstmt.setDouble(10,salario);
          pstmt.executeUpdate();
          con.commit();
          pstmt.close();
          con.close();
          setText("Dados de Retorno inseridos com sucesso no banco de dados
local!");
          waitMessage(2 * 1000);
          dispose();
         }
        }
        catch(Exception e)
        {
         setText(String.valueOf(e));
        }
      }
      else
      {
        try
        {
          Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
          String url   = "jdbc:odbc:" + "rh";
          con = DriverManager.getConnection(url, "", "");
          stmt =  con.createStatement();
          String strSQL = "SELECT a.Nome, a.cd_banco, a.nu_conta,
b.tot_horas_trab, b.tot_faltas, b.tot_Atestados, b.tot_horas_extras FROM
Servidor a, Frequencia b WHERE  ano = '" + anoEscolhido  + "' and mes='" +
mesEscolhido +"' and a.cd_servidor =b.cd_servidor";
          rs =  stmt.executeQuery(strSQL);
          String txtSaida = "";
          while(rs.next())
           {
             txtSaida = txtSaida + rs.getString(1) + ";" + rs.getString(2)
+ ";" + rs.getString(3) + ";" + rs.getString(4) + ";" + rs.getString(5) +
";"+ rs.getString(6) + ";" + rs.getString(7) + ";";
           }
          stmt.close();
          Message mensagem = new Message("Retorna");
          mensagem.setArg("dados",txtSaida);
          handleMessage(mensagem);
          con.close();
        }
        catch(Exception e)
        {
          setText(String.valueOf(e));
        }
      }
    }


     public void Envia(Message msg)
     {
          String destino = String.valueOf(msg.getArg("destino"));
          String mes = String.valueOf(msg.getArg("mes"));
          mesEscolhido = mes;
          String ano = String.valueOf(msg.getArg("ano"));
          anoEscolhido = ano;
```

```
      try
      {
         dispatch(new URL (destino));
      }
      catch(Exception e)
      {
         setText("Erro ao enviar o Agente ao Host Remoto");
      }
    }


    public void Recebe(Message msg)
    {
      dadosRetorno = String.valueOf(msg.getArg("dados"));
      try
      {
         dispatch(new URL (home));
      }
      catch(Exception e)
      {
         setText("Erro ao enviar o Agente de volta a Origem");
      }
    }


    public boolean handleMessage(Message msg) {
        if (msg.sameKind("Envia")) {
           Envia(msg);
        } else if (msg.sameKind("Retorna")) {
           Recebe(msg);
        } else {
            return false;
        }
        return true;
    }


       public void dialog(Message msg) {
        // check and create a dialog box
        if (my_dialog == null) {
            my_dialog = new MyDialog(this);
            my_dialog.pack();
            my_dialog.resize(my_dialog.preferredSize());
        }

        // show the dialog box
        my_dialog.show();
    }
}


class MyDialog extends Frame {

    private Banco aglet  = null;
    private Choice filial      = new Choice();
    private Choice ano          = new Choice();
    private Choice mes          = new Choice();
    private Label  diretoria    = new Label();
    private Label  anoL         = new Label();
    private Label  mesL         = new Label();
    private TextField msg       = new TextField(18);
```

```
    private Button go           = new Button("Enviar");
    private Button close        = new Button("Fechar");

    MyDialog(Banco aglet) {
        this.aglet = aglet;
        layoutComponents();
    }

    private void layoutComponents() {
        aglet.message ="";

        // Layouts components
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cns = new GridBagConstraints();
        setLayout(grid);

        cns.weightx = 0.5;
        cns.ipadx = cns.ipady = 5;
        cns.fill = GridBagConstraints.HORIZONTAL;
        cns.insets = new Insets(5,5,5,5);

        cns.weightx = 1.0;
        cns.gridwidth = GridBagConstraints.REMAINDER;
        grid.setConstraints(filial, cns);
        add(diretoria);
        add(filial);


        cns.gridwidth = GridBagConstraints.REMAINDER;
        cns.fill = GridBagConstraints.BOTH;
        cns.weightx = 1.0;
        cns.weighty = 1.0;
        cns.gridheight = 2;
        grid.setConstraints(ano, cns);
        add(anoL);
        add(ano);

        cns.gridwidth = GridBagConstraints.REMAINDER;
        cns.fill = GridBagConstraints.BOTH;
        cns.weightx = 1.5;
        cns.weighty = 1.5;
        cns.gridheight = 3;
        grid.setConstraints(mes, cns);
        add(mesL);
        add(mes);

        cns.weighty = 0.0;
        cns.fill = GridBagConstraints.NONE;
        cns.gridheight = 1;

        Panel p = new Panel();

        grid.setConstraints(p, cns);
        add(p);
        p.setLayout(new FlowLayout());
        p.add(go);
        p.add(close);
        diretoria.setText("Filial:");
        anoL.setText("Ano:");
        mesL.setText("Mês:");
        filial.addItem("São José dos Campos");
```

```
        filial.addItem("Florianópolis");
        filial.addItem("Caxias do Sul");
        filial.addItem("Treze Tilhas");
        filial.addItem("São Bento do Sul");
        filial.addItem("Salvador");
        filial.addItem("Vitória");
        ano.addItem("2002");
        ano.addItem("2001");
        ano.addItem("2000");
        mes.addItem("Jan");
        mes.addItem("Fev");
        mes.addItem("Mar");
        mes.addItem("Abr");
        mes.addItem("Mai");
        mes.addItem("Jun");
        mes.addItem("Jul");
        mes.addItem("Ago");
        mes.addItem("Set");
        mes.addItem("Out");
        mes.addItem("Nov");
        mes.addItem("Dez");
    }



    public boolean handleEvent(Event ev) {
        if (ev.id == Event.WINDOW_DESTROY) {
            hide();
            return true;
        }
        return super.handleEvent(ev);
    }

    public boolean action(Event ev, Object obj) {
        String destino = "";
        String aURL[] = new String[7] ;
        aURL[0] = "atp://dial:100";
        aURL[1] = "atp://dial:200";
        aURL[2] = "atp://dial:300";
        aURL[3] = "atp://dial:400";
        aURL[4] = "atp://dial:500";
        aURL[5] = "atp://dial:600";
        aURL[6] = "atp://dial:700";
        if (ev.target == go) {
            String filialConsultada = filial.getSelectedItem();
            int item = filial.getSelectedIndex();
            destino = aURL[item];
            String anoConsultado = ano.getSelectedItem();
            String mesConsultado = mes.getSelectedItem();
           Message mensagem = new Message("Envia");
            mensagem.setArg("destino",destino);
            mensagem.setArg("ano",anoConsultado);
            mensagem.setArg("mes",mesConsultado);
            aglet.handleMessage(mensagem);
        } else if (ev.target == close) {
            hide();
        } else {
            return false;
        }
        return true;
```

```
        }
}
```

9 Referências Bibliográficas

ARTHURSON Johan et al ***A Platform for Secure Mobile Agents***, Abril 1997. http://citeseer.nj.nec.com/arthursson97platform.html

BIGUS Jennifer; BIGUS Joseph ***Constructing Intelligent Agents with Java***; Willey Computer Publishing, 1998.

BRENNER Walter at al ***Inteligent Software Agents***; Springer-Verlag Press,1998.

CHESS David at al ***IBM Research Report:*** *Itinerant Agents for Mobile Computing*. Zurich: Switzerland, 1995.

FARMER at al ***Security for Mobile Agents: Authentication and State Appraisal***, Novembro 2001. http://citeseer.nj.nec.com/farmer96security.html

HAGIMONT, Daniel; ISMAIL Leila ***A Protection Scheme for Mobile Agents on Java***, Dezembro, 2000. http://citeseer.nj.nec.com/43375.html

KARJOTH, Güinter at al ***A Security Model for Aglets***, Novembro, 2001. http://citeseer.nj.nec.com/karjoth97security.html

LANGE, Danny B., OSHIMA Mitsuru ***Programming and Deploying Java Mobile Agents with Aglets***. Massachusetts: Addison-Wesley, 1998. p. 171-185.

OAKS, Scott **Segurança de Dados em Java**. Rio de Janeiro: Ed. Ciência Moderna, 1999. 1ª Edição P.171-185.

OAKS, Scott ***Java Security***. Califórnia: Nutshell Hand Books, 2001. 2ª Edição.

SANDER Tomas, TSCHUDIN Christian F. **Protectiong Mobile Agents Against Malicious Hosts**. Submitted to the fourthcoming LNCS on "Mobile Agent Security", November 11, 1997.

SOARES, Luiz Fernando at al **Redes de Computadores**: das LANs, MANs e WANs as Redes ATM. Rio de Janeiro: Editora: CAMPUS, 1998. 2ª Edição.

SPYMAN, Hacking **Manual Completo do Hacker – Edição 2001**. Rio de Janeiro: Book Express, 2001.

STALLINGS, William ***Cryptografy and Network Security:*** *principals and practice*. New Jersey: Prentice-Hall, 1998. P.1-92.

VALDO, Clayton A. **CuBe**: Um Sistema para Integrar Comércio Eletrônico e ERP Através de Agentes Móveis. Dissertação de Mestrado submetida a Universidade Federal de Santa Catarina, Setembro, 2000

WALSH T. at al **Security and Reliability in Concordia.** Proceedings of the 31st Hawaii International Conference on Systems Sciences, VII:44-53, January 1998.

10 Índice