MICHELLE SILVA WANGHAM

ESQUEMA DE SEGURANÇA PARA AGENTES MÓVEIS EM SISTEMAS ABERTOS

FLORIANÓPOLIS
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESQUEMA DE SEGURANÇA PARA AGENTES MÓVEIS EM SISTEMAS ABERTOS

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutora em Engenharia Elétrica.

MICHELLE SILVA WANGHAM

Florianópolis, dezembro de 2004.

ESQUEMA DE SEGURANÇA PARA AGENTES MÓVEIS EM SISTEMAS ABERTOS

Michelle Silva Wangham

'Esta Tese foi julgada adequada para a obtenção do título de Doutora em Engenharia Elétrica, Área de Concentração em *Sistemas de Informação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Denizar Cruz Martins, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Presidente

Prof. Ricardo Dahab, Ph.D.

Prof. Edgard Jamhour, Dr.

Prof. Ricardo Felipe Custódio, Dr.

Prof. Ricardo José Rabelo, Dr.

Prof. Carlos Barros Montez, Dr

Sempre que me bate o desânimo, a desesperança e o temor,
procuro me desvencilhar destes sentimentos, lembrando que,
lá no final, está um sonho muito maior que o meu.
Está o sonho de Deus. **Ivna Sá**

Aos meus mestres e pais Eva e Jura
e ao homem da minha vida
Luiz Magno…

AGRADECIMENTOS

À ti Senhor, que tens guiado os meus passos e me trouxeste em paz até aqui, o meu sincero agradecimento. "*Por abrigo tenho o teu amor... És meu companheiro, meu melhor amigo.*"..."*Pois Tu és o meu amparo o meu refúgio, és alegria de minh'alma...*". Por esta tese defendida, eu dou Glória ao Pai, ao Filho e ao Espírito Santo. Amém.

As dificuldades e desafios enfrentados neste doutorado se tornaram mais leves graças ao amor, suporte e paciência do meu amado Luiz. Homem da minha vida, obrigada por dividir comigo as tristezas e alegrias nestes longos anos de doutorado. Esta vitória também é sua!

Aos meus maiores incentivadores e mestres, meus pais, Evani e Jurandy Wangham, que me ensinaram a amar a profissão que escolhi, e aos meus irmãos, Danielle e Paulo, que sempre me encheram de amor, carinho, mimos e ensinamentos, requisitos fundamentais para a realização deste sonho.

Agradeço ao Prof. Joni da Silva Fraga, não só pela orientação neste trabalho, mas também pela amizade, críticas, incentivos e discussões que me fizeram amadurecer como pessoa e pesquisadora e me apaixonar pela academia.

Agradeço de forma especial a família GOU/UFSC (Grupo de Oração Universitário), por todo amor, amizade, orações, reuniões, encontros e tantos momentos especiais que me fizeram descansar nos braços do Pai e experimentar do Seu amor. Meus amigos pela fé, a amizade de vocês, especialmente nos momentos finais deste trabalho, foram essenciais para que de pé e com Deus eu superasse todas as dificuldades. Agradeço ainda a todos que compartilham deste mesmo sonho de amor, fazendo parte do Projeto Universidades Renovadas, obrigada por seus pães e peixes e por participarem da construção da civilização do amor. "*Pai por tudo que vivemos, obrigada! Por tudo que viveremos, sim!*"(dos Santos, 2004)

Aos colegas e parceiros na implementação deste trabalho, Ricardo Schmidt, Rafael J. Deitos, Galeno A. Jung e Elizabeth Fernandes, cujas contribuições enriqueceram este trabalho, o meu sincero agradecimento. Foi gratificante trabalhar com vocês.

Agradeço à minha família em Florianópolis, Karina, Cris, Magali, Felipe, Ivair, Hallthmann, Léo, Tati e Rafael, que contribuíram e ainda contribuem para que a qualidade de vida em Florianópolis seja melhor ainda do que a noticiada nos jornais. Aos meus amigos de Belém e espalhados pelo mundo a fora, na estrada tão longe e na alma tão perto, obrigada por todo carinho e amizade de vocês.

Aos professores e colegas do DAS que conheci e convivi ao longo desses anos, em especial ao Alisson, Cássia, Emerson, Fabinho, Fábio Pinga, Fabio Baiano, Jerusa, Pat Pena, Vallim, Sumar, Zezão e Lu, obrigada pelas contribuições e por tornarem o nosso ambiente de trabalho mais agradável e divertido.

Finalmente, agradeço aos professores e funcionários do Programa de Pós-Graduação em Engenharia Elétrica, por nos proporcionarem um curso de doutorado de qualidade.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

ESQUEMA DE SEGURANÇA PARA AGENTES MÓVEIS EM SISTEMAS ABERTOS

Michelle Silva Wangham

Dezembro/2004

Orientador: Joni da Silva Fraga
Área de Concentração: Sistema de Informação
Palavras-chave: Segurança, Agentes móveis, Sistemas abertos
Número de Páginas: xviii + 187

O paradigma de agentes móveis vem sendo utilizado em sistemas distribuídos, principalmente, devido a sua flexibilidade proveniente da noção de mobilidade. A capacidade para mover agentes em um sistema aberto permite o desenvolvimento de serviços e aplicações mais flexíveis e dinâmicos quando comparado com o modelo cliente-servidor. Apesar das suas vantagens, a tecnologia de agentes móveis introduz novas ameaças de segurança ao sistema que, muitas vezes, impedem a sua ampla aceitação. Esta tese propõe um esquema de segurança para aplicações baseadas em agentes móveis em sistemas abertos (chamado *MASS*), composto de técnicas de prevenção e de detecção, que visam prover segurança para o canal de comunicação, para as plataformas de agentes e para os próprios agentes. Para ser corretamente implantado em sistemas abertos, este esquema combina os aspectos da segurança com questões de portabilidade, interoperabilidade, escalabilidade, compatibilidade, simplicidade e desempenho. As técnicas disponíveis no *MASS* estão baseadas na infra-estrutura SPKI, no conceito de Federação SPKI e em protocolos criptográficos. Para atender às necessidades específicas de aplicações, este esquema é flexível de modo que o mesmo pode ser especializado através da seleção de um subconjunto de mecanismos. Ainda neste trabalho, um protótipo foi definido, implementado e integrado a uma aplicação distribuída de forma a comprovar a sua flexibilidade e viabilidade de uso em sistemas abertos.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

SECURITY SCHEME FOR MOBILE AGENTS IN OPEN SYSTEMS

Michelle Silva Wangham

December/2004

Advisor: Joni da Silva Fraga
Area of Concentration: Information Systems
Key words: Security, Mobile agents, Open systems
Number of Pages: xviii + 187

The mobile agent paradigm has been used in distributed systems due mainly to its flexibility, which comes from the mobility notion. The capability to move agents within an open system allows the development of more flexible and dynamic services and applications, if compared to the client-server model. Despite its advantages, the mobile agent technology introduces new security threats into the system. These threats frequently prevent its wide acceptance in the field. This thesis proposes a security scheme to applications based on mobile agents in open systems (called *MASS*). The *MASS* Scheme is composed of prevention and detection techniques that aim at providing security to (1) the communications channel, (2) agents platforms, and (3) agents themselves. For the purposes of being implemented in open systems appropriately, this scheme combines the security aspects with portability, interoperability, scalability, compatibility, simplicity and performance. The techniques available in *MASS* are based on the SPKI infrastructure, in the concept of SPKI Federation, and in cryptographic protocols. So as to meet the needs of specific applications, this scheme is flexible in a way that it can become specialized, through the selection of a subset of mechanisms. Also, in the present study, a prototype was defined, implemented and integrated into a distributed application in order to demonstrate its use flexibility and suitability in open systems.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A expansão das redes de computadores no mundo dos negócios tem estimulado diversas pesquisas em sistemas distribuídos de larga escala. Em resposta a este sucesso e ao interesse despertado tanto pelas empresas quanto pelos centros de pesquisa, novas técnicas, linguagens e paradigmas surgiram para facilitar a criação de aplicações para diversas áreas. Um paradigma que tem recebido grande destaque e interesse em pesquisas é o que incorpora a noção de agente móvel. Um agente móvel pode ser definido como um agente de *software* que migra de um sítio para outro em uma rede heterogênea em nome de um indivíduo ou organização, que se executa de maneira autônoma no seu destino. Um agente visita um sítio para executar partes de seu programa e pode, enquanto trabalha no sentido de cumprir seus objetivos, interagir com outros agentes residentes no mesmo sítio ou interagir com agentes em outros sítios. A utilização de agentes móveis requer que as máquinas onde estes agentes serão executados possuam um ambiente computacional específico: uma plataforma de agentes.

A tecnologia de agentes móveis vem se constituindo como um paradigma chave para ambientes distribuídos, abertos e dinâmicos. Esta tecnologia emergente promete facilidades para o desenvolvimento de projetos, para a implementação e para a manutenção de sistemas distribuídos. Entre as suas vantagens, pode-se destacar que: os agentes móveis podem reduzir o tráfego da rede, fornecer meios efetivos para diminuir a latência da rede, ajudar as interfaces e os serviços de sistemas remotos a se adaptarem dinamicamente, reduzir a dependência sobre a disponibilidade constante de conexões de redes caras e frágeis e, principalmente, através de sua habilidade para operar assíncrono e autonomamente, ajudar a construir serviços adaptados para sistemas distribuídos de larga escala.

A incorporação da noção de agentes móveis nestes sistemas de larga escala exige que as plataformas de agentes sigam padrões que visem garantir a interoperabilidade entre plataformas de diferentes fabricantes. A garantia desta interoperabilidade configura-se como um requisito fundamental para atender integralmente às necessidades do mercado aberto de serviços da Internet, por exemplo.

Já são muitos os domínios de aplicação que procuram usufruir dos benefícios dos agentes móveis, porém o conjunto das ferramentas para a construção de aplicações que incorporem a noção de mobilidade de código ainda é pequeno quando comparado às aplicações baseadas no modelo cliente-servidor. A aplicabilidade da mobilidade de código e as evidências experimentais ainda estão sendo verificadas principalmente devido às preocupações com a segurança, com a robustez, com a interoperabilidade e com o desempenho.

Para se ter uma idéia do futuro promissor do paradigma de agentes móveis, as possibilidades de seu uso em alguns domínios de aplicação estão descritos a seguir (Picco, 1998):

- **Consulta de Informação Distribuída**. As aplicações de consulta de informações distribuídas envolvem a busca de informações em fontes normalmente dispersas na rede. As fontes de informação a serem "visitadas"podem ser definidas de modo estático ou determinadas dinamicamente durante a execução de uma requisição de consulta. Várias aplicações na Internet se encaixam neste domínio de busca de informação. Os objetivos destas buscas podem variar, por exemplo, do conjunto de publicações de um dado autor até o endereço de todos os *hosts* conectados em uma rede qualquer. A mobilidade de código neste domínio pode trazer grandes benefícios às aplicações. Ao transferir o código para a fonte de informação, por exemplo, é possível obter ganhos em eficiência uma vez que a consulta e a filtragem das informações serão realizadas localmente na própria base de informação. Este domínio de aplicação foi um dos primeiros domínios explorados com o uso do paradigma de agentes móveis.

- **Serviços de Telecomunicações**. Atividades de gerenciamento e a contabilização de alguns serviços de telecomunicações avançados, como a vídeo-conferência, vídeo-sob-demanda e tele-conferência, necessitam de um *middleware* especializado que forneça mecanismos para reconfiguração dinâmica e customização (*customizing*) de usuários. A mobilidade de código pode facilitar a implementação destas atividades ao desenvolver componentes com funções de gerência de *setup*, sinalização e de serviços de apresentação para uma vídeo-conferência (Magedanz *et al.*, 1996). Uma classe avançada de serviços de telecomunicações é a que suporta usuários móveis. Neste caso, componentes autônomos podem prover suporte para usuários com conexões temporárias devido à capacidade de operação assíncrona dos agentes móveis. Um agente móvel é capaz de monitorar uma fonte de informações sem ficar dependente do local onde este foi originado.

- **Gerenciamento e Cooperação de Atividades do Tipo *Workflow***. Aplicações de *Workflow* e *Groupware* possibilitam a cooperação de pessoas e a integração de ferramentas na execução de uma aplicação automatizada. O fluxo de controle dessas aplicações é determinado por um roteiro de atividades delegadas a cada participante (grupos de funcionários e/ou departamento), e deve ser executado dentro do período estabelecido no roteiro. Estas aplicações geralmente são modeladas, sendo as atividades representadas como entidades autônomas, que no decorrer do tempo, circulam

entre as entidades envolvidas no *workflow*. A mobilidade de código parece ser perfeitamente adequada para o desenvolvimento deste tipo de aplicação, já que o próprio paradigma de agente móvel permite incluir o estado das atividades nas entidades móveis. Uma aplicação de revisão de textos, por exemplo, poderia compor as entidades, como o texto a ser revisado, juntamente com o estado das revisões já realizadas e a próxima operação a ser executada.

- **Comércio Eletrônico**. Aplicações de comércio eletrônico permitem a realização de negócios em um ambiente de rede. O ambiente dessa aplicação geralmente é composto por diversas entidades comerciais independentes e competidoras entre si. Assim, a segurança é um dos fatores mais importantes para o sucesso e aceitação da aplicação. A realização de um negócio envolve várias etapas de trocas de mensagens com entidades remotas e pode exigir acesso imediato a recursos remotos, como informações que são atualizadas constantemente, como os índices da bolsa de valores. A mobilidade de código é uma promessa de reestruturação do comércio eletrônico, uma vez que os agentes "consumidores", incorporando as intenções de seus proprietários, pesquisarão e negociarão o melhor preço e prazos antes de efetuar uma compra.

Apesar da mobilidade de código permitir o desenvolvimento de serviços e aplicações mais flexíveis e dinâmicas, se comparada ao paradigma cliente-servidor, a sua aceitação ainda é prejudicada devido, principalmente, às questões de segurança. As preocupações dos desenvolvedores de sistemas, dos administradores de redes e dos gerentes de sistemas de informação com a segurança aumentam quando o conteúdo executável (agente) e o seu estado de execução são movidos entre diferentes ambientes computacionais. O problema se agrava quando o território da aplicação abrange um sistema aberto e de larga escala como a Internet, herdando assim não só as facilidades, mas também as vulnerabilidades inerentes à grande rede mundial. Neste cenário, ataques variados podem ser cometidos por qualquer elemento do sistema (plataforma de agentes ou agentes) ou até mesmo por um elemento externo ao sistema, como um usuário espionando o conteúdo dos pacotes transmitidos na rede.

Uma vez resolvidas ou minimizadas as questões referentes a segurança e se uma coleção de mecanismos de segurança estiver disponível para tratar de possíveis riscos, a tecnologia de agentes móveis se tornará uma excelente opção para o desenvolvimento de soluções proveitosas e inovadoras para problemas existentes e para encontrar novos domínios de aplicação que irão se beneficiar com esta tecnologia.

1.1 Motivação

A segurança de informações protege as aplicações contra uma ampla gama de ameaças a fim de assegurar a continuidade dos negócios, de minimizar prejuízos e maximizar o retorno

de investimentos, possibilitando novas oportunidades comerciais. Algumas questões de segurança têm sido estudadas pela comunidade científica de sistemas distribuídos já há algum tempo, entretanto, os mecanismos e tecnologias desenvolvidos, por exemplo, para a comunicação segura, para a autenticação e para a autorização, precisam ser aprimorados para que a mobilidade de código seja levada em conta. Algumas ameaças de segurança são específicas do paradigma de agentes móveis, logo, técnicas específicas precisam ser desenvolvidas para garantir a segurança neste paradigma.

Além disso, os mecanismos disponíveis hoje para contornar os riscos desta tecnologia não cobrem, com eficiência, todas as ameaças existentes, relativas ao paradigma de agentes móveis, além de introduzirem restrições de desempenho, de tal maneira que inviabilizam o uso de agentes móveis para certas aplicações. Em face das características do paradigma de agentes móveis e das classes de ameaças a que este está exposto, os mecanismos de segurança podem ser direcionados à segurança do canal de comunicação, da plataforma e dos agentes móveis.

A **segurança do canal de comunicação** é uma preocupação inerente aos sistemas distribuídos, independentemente da adoção ou não do paradigma de agentes móveis, razão pela qual diversas soluções já se encontram descritas na literatura.

Algumas soluções para a **segurança das plataformas de agentes móveis** já produzem resultados satisfatórios para alguns domínios de aplicação. Alguns mecanismos implantados nas plataformas comerciais e acadêmicas de agentes móveis apresentam uma certa eficiência, e são aceitos, de um modo geral, na literatura. Entre estas soluções pode-se destacar a combinação da técnica de domínios isolados de execução com a de assinaturas digitais, conforme definido no modelo de segurança do Java 2. Porém, estas técnicas precisam ser aprimoradas, em especial, quando agentes com itinerários livres são considerados e quando estes agentes percorrerão um sistema aberto e de larga escala. A confiança da plataforma em um agente não depende só da verificação da autenticidade do proprietário do agente mas também da confiança nas plataformas visitadas pelo agente, já que um agente móvel pode se tornar malicioso em virtude do seu estado ter sido corrompido por plataformas visitadas anteriormente. Garantir a segurança e a confiabilidade dos agentes com itinerários livres, já que não se conhece previamente o seu destino e as plataformas visitadas, não é tarefa fácil e ainda não possui uma solução efetiva.

Para a proteção de plataformas de agentes contra ataques de outras plataformas, protocolos de autenticação mútua, usando técnicas de verificação de assinaturas digitais (por exemplo, as implementadas no protocolo SSL), são normalmente empregados em plataformas comerciais e acadêmicas. Porém, somente a autenticação mútua não é suficiente; é preciso associar a estes protocolos de autenticação listas de plataformas autorizadas e políticas de isolamento de plataformas maliciosas para minimizar a possibilidade de violações de segurança.

A **segurança dos agentes móveis** contra as ameaças de plataformas maliciosas é um pro-

blema ainda não resolvido, já que, durante a interpretação de um agente móvel, o ambiente computacional pode ter total acesso ao código e ao estado do agente. Técnicas de prevenção e detecção para a segurança de agentes móveis têm sido apresentadas na literatura, porém, estas técnicas ainda não podem ser consideradas práticas e flexíveis, em especial quando os agentes precisam percorrer vários sítios em uma rede de larga escala. Devido a isto, são poucas as plataformas de agentes móveis comerciais e acadêmicas que se preocupam com as ameaças decorrentes de plataformas maliciosas.

As diversas ameaças às quais as aplicações baseadas em agentes móveis são suscetíveis e as limitações das técnicas de proteção comprovam a necessidade de um esquema, composto por um conjunto de mecanismos de prevenção e de detecção, que garanta a segurança em sistemas de agentes móveis. Este esquema de segurança deve oferecer o suporte adequado para aplicações em sistemas abertos que anseiam por escalabilidade, portabilidade, interoperabilidade e mobilidade.

Diante da constatação da necessidade do desenvolvimento de um esquema de segurança flexível e eficiente e da preocupação com aspectos específicos da segurança de agentes móveis em sistemas distribuídos e abertos, algumas questões de pesquisa surgiram indicando os desafios a serem enfrentados. São elas:

- Será possível, a partir de um modelo de agentes de múltiplos saltos (*multi-hop*) e de um ambiente heterogêneo e distribuído de plataformas, conceber um esquema de autenticação e de autorização descentralizado e dinâmico e implementar políticas de segurança para sistemas abertos? Como um agente móvel *multi-hop* pode ser autenticado por uma plataforma de agentes? Como expressar de forma flexível e escalável as credenciais que um proprietário pode fornecer ao seu agente?
- É possível prevenir e/ou detectar ataques de plataformas maliciosas? Como minimizar os riscos de plataformas maliciosas contra os agentes móveis *multi-hop*? É possível combinar mecanismos de segurança para prevenir e/ou detectar a revelação e modificação não autorizada de dados carregados pelo agente? Como oferecer e suportar de forma flexível um conjunto de mecanismos compatíveis para construção de aplicações de agentes móveis protegidos? Como estabelecer e fiscalizar um comportamento correto que deve ser imposto às plataformas de agentes em um ambiente como a Internet?
- É possível avaliar os mecanismos de segurança propostos sem conceber um protótipo? É possível construir um protótipo flexível, eficiente e de fácil configuração e uso para o projeto e implementação de aplicações seguras baseadas em agentes móveis? Quais tecnologias devem ser utilizadas para construir o protótipo do esquema de segurança que ofereça as características desejadas? Como minimizar as degradações de desempenho provocadas pelo uso dos mecanismos de proteção? Como avaliar os riscos a que uma aplicação baseada em agentes móveis, que usufrui do esquema de segurança, está sujeita?

Estas questões de pesquisa motivaram a presente tese e orientaram a formulação de seus objetivos descritos na seção a seguir.

1.2 Objetivos da Tese

O objetivo geral desta tese é desenvolver um esquema de segurança para aplicações baseadas em agentes móveis em sistemas abertos. Para isto, um esquema de autenticação e de autorização para a proteção das plataformas de agentes móveis e dos recursos do sistema, entre eles os próprios agentes, é proposto. Este esquema prevê ainda técnicas de prevenção e de detecção com o intuito de contornar ou minimizar ataques de plataformas maliciosas. Para ser corretamente implantado em sistemas abertos, este esquema combina os aspectos da segurança com questões de portabilidade, interoperabilidade, escalabilidade, desempenho e flexibilidade.

Baseado em um modelo de agentes móveis com itinerário livre e *multi-hop* e visando atender ao objetivo geral desta proposta, os seguintes objetivos específicos foram perseguidos:

- desenvolver um esquema de autenticação e de autorização flexível e descentralizado para plataformas de agentes móveis, com mecanismos de prevenção e de detecção, adequado para sistemas abertos;
- elaborar mecanismos de prevenção e de detecção para contornar e minimizar os riscos que os agentes móveis estão suscetíveis quando estes estão se executando em plataformas maliciosas; e, desenvolver técnicas que permitam a identificação de plataformas maliciosas;
- definir e implementar um protótipo que inclua o esquema de segurança proposto, bem como as técnicas de fiscalização e detecção de plataformas maliciosas;
- integrar e adequar o protótipo a uma aplicação distribuída e híbrida, baseada em agentes móveis e agentes estacionários, de empresas virtuais.

1.3 Organização do Texto

O texto da tese é uma reflexão sobre as diversas etapas que foram cumpridas durante o doutoramento e está dividido em seis capítulos. Neste primeiro capítulo foi descrito o contexto geral em que o trabalho está inserido, destacando as questões de pesquisa, os objetivos da proposta de tese e a motivação para a escolha do tema.

O capítulo 2 analisa os principais conceitos relacionados com segurança de informação. São discutidos os objetivos da segurança da informação, as técnicas nas quais os mecanismos de segurança estão baseados, os modelos de segurança e as abordagens de implementação dos mecanismos de autenticação e autorização.

O principais conceitos que envolvem o paradigma de agentes móveis são apresentados no Capítulo 3, tendo como base um arcabouço conceitual para códigos móveis e um modelo conceitual comum de agentes móveis definido pela OMG. As vantagens e desvantagens deste paradigma também são discutidas.

No capítulo 4, a problemática da segurança é discutida a partir da análise das ameaças de segurança encontradas em sistemas baseados em agentes móveis e das limitações das técnicas de proteção. Uma visão geral dos mecanismos que compõem o esquema de segurança proposto nesta tese é apresentada, descrevendo os objetivos, as premissas e os suportes necessários do esquema proposto.

O Capitulo 5 apresenta uma revisão da literatura que trata do problema da proteção das plataformas de agentes móveis e descreve o esquema de autenticação e de autorização proposto e que visa contornar as ameças de agentes maliciosos. No final do capítulo, o esquema proposto é comparado com os trabalhos relacionados.

Uma revisão das principais abordagens e técnicas que visam proteger os agentes móveis contra plataformas maliciosas é apresentada no Capítulo 6 e o esquema proposto que visa minimizar os riscos que os agentes móveis estão sujeitos é descrito. Ainda neste capítulo, uma comparação dos trabalhos relacionados com o esquema proposto é apresentada e discutida.

O capítulo 7 descreve o protótipo do esquema de segurança, apresenta as escolhas tecnológicas que compõem a arquitetura do protótipo e analisa os testes de *software* e de desempenho realizados. Por fim, a integração do protótipo a uma aplicação distribuída é discutida de forma a permitir a análise da potencialidade do protótipo desenvolvido.

Finalmente, no capítulo 8 são discutidos os resultados obtidos e a sua relevância no contexto da segurança de agentes móveis em sistemas abertos. São apresentadas, como sugestão, algumas propostas para trabalhos futuros nesta área.

Capítulo 2

Segurança em Sistemas Informáticos Distribuídos

As dimensões que as redes de computadores e os sistemas distribuídos estão assumindo têm sido determinantes para uma crescente dependência das corporações em relação a seus sistemas de informação. Em ambientes de larga escala, por onde circulam informações importantes para as corporações e, muitas vezes, sigilosas, manter a segurança das informações é uma necessidade vital. As preocupações com a segurança em sistemas informáticos crescem continuamente devido ao aumento do uso da Internet e da disseminação de computadores em ambientes empresariais, governamentais e de ensino. Esta preocupação é agravada pelos diversos registros de incidentes de segurança provocados por intrusos ou por funcionários insatisfeitos (CSI/FBI, 1999) e pelos numerosos relatos de vulnerabilidades em todas as classes de ferramentas e suportes de um sistema, inclusive nos *softwares* de segurança (Schneier, 2000). Complementando, a proteção física de computadores, que contêm as informações das corporações, é um problema de muitos anos e dificilmente é concretizada nas empresas e nas instituições governamentais e de ensino. A conjunção desses fatores resulta em um cenário em que a segurança computacional em sistemas distribuídos se mostra importante, porém difícil e complexa.

Este capítulo apresenta alguns conceitos sobre segurança computacional em sistemas distribuídos, necessários para um bom entendimento desta tese. Um texto mais completo e aprofundado sobre segurança computacional pode ser encontrado em Bishop (2003). Neste capítulo, serão revisados os conceitos fundamentais, os objetivos de segurança, as técnicas nas quais os mecanismos de segurança estão baseados e as abordagens de implementação dos mecanismos de autenticação e de autorização. No Apêndice A, o modelo de segurança Java é descrito, analisando-o como suporte em *middleware* para segurança.

2.1 Conceitos Fundamentais

2.1.1 Segurança Computacional

O conceito de segurança [1] em um sistema informático é identificado como a capacidade de assegurar a prevenção ao acesso e à manipulação ilegítima da informação, ou ainda, de evitar a interferência indevida na sua operação normal (ISO/IEC, 1999). Ou seja, a definição convencional de segurança computacional requer que o sistema garanta três propriedades fundamentais (Landwehr, 2001; Bishop, 2003):

- **confidencialidade**: garante que as informações serão lidas somente por usuários autorizados; a violação da confidencialidade é chamada de **revelação não-autorizada** ou vazamento de informação;
- **integridade**: garante a consistência dos dados, em particular, prevenindo a criação não autorizada e a alteração ou destruição dos dados, provocados por faltas intencionais ou acidentais; a violação da integridade é chamada de **modificação não autorizada**;
- **disponibilidade**: garante que usuários legítimos não terão o acesso indevidamente negado a informações e recursos; a violação da disponibilidade é chamada **negação de serviço**;

Recentemente, tem sido comum adicionar outras duas propriedades de segurança a esta definição:

- **autenticidade**: garante que cada principal é quem diz ser; a violação da autenticidade é chamada de **personificação**;
- não-repudiação: garante que uma terceira parte pode ser convencida de que uma transação particular ou evento ocorreu ou não; a violação dessa propriedade é chamada de **repúdio**.

Na prática, a concepção de sistemas em que essas violações de segurança são totalmente evitadas é muito difícil. Os problemas de confinamento definidos por Lampson (1973) são a prova da dificuldade de evitar a revelação não autorizada. Todo o sistema em que a concepção está baseada no compartilhamento de recursos apresenta possibilidades sutis e não esperadas para a transferência de informação não autorizada entre duas entidades ou para a indisponibilidade de um serviço.

[1] Trata-se aqui da chamada segurança de dados (*data security*) ou de informação em oposição à chamada segurança (*safety*) ou correção em sistemas críticos.

2.1.2 Ameaças, Vulnerabilidades, Ataques e Riscos

Os conceitos fundamentais apresentados, a seguir, estão de acordo com Ford (1994); Landwehr (2001).

Vulnerabilidades são características indesejáveis (faltas ou defeitos) em sistemas que podem ser exploradas para concretizar uma violação de segurança. Vale ressaltar que as probabilidades de violações em um sistema podem ser minimizadas através da identificação e remoção de vulnerabilidades existentes no sistema. **Riscos** são uma medida do custo de uma vulnerabilidade que incorpora a probabilidade de ocorrer uma violação de segurança.

Uma **ameaça** a um sistema computacional é a possibilidade de provocar violações de segurança em um sistema; consiste em qualquer circunstância ou um conjunto de ações que possa fornecer algum potencial de violação de segurança. Uma vez concretizada, a ameaça pode produzir efeitos indesejáveis sobre os dados e/ou recursos do sistema.

Ataque é a concretização de ameaças, podendo ser associado com ações que envolvem a exploração de certas vulnerabilidades, visando uma ou mais violações de segurança. Os ataques são classificados como: **passivos**, os que tentam quebrar somente a confidencialidade dos dados (revelação não autorizada da informação); e **ativos** que envolvem ações no sentido de concretizar modificações não autorizadas e a negação de serviço, afetando a integridade e/ou a disponibilidade das informações.

2.1.3 Políticas, Modelos e Mecanismos de Segurança

Os conceitos de políticas de segurança, modelos de segurança e mecanismos de segurança são freqüentemente confundidos na literatura. Visando eliminar possíveis dúvidas, esses conceitos serão apresentados, a seguir, tendo em vista o contexto deste trabalho.

A **política de segurança** de um sistema é o conjunto de regras e práticas que estabelecem os limites de operação dos usuários do sistema e que determinam a maneira pela qual as informações e os recursos são administrados, protegidos e distribuídos no interior de um sistema específico (ISO/IEC, 1999). A noção de política de segurança pode ser destacada em três ramos distintos (Nicomette, 1996): políticas de segurança física, política de segurança administrativa e política de segurança lógica. A primeira se ocupa com tudo que se refere à situação física do sistema a proteger. Neste ramo, em particular, são definidas as medidas contra a violação de segurança física provocadas por incêndio, catástrofes naturais etc. A política administrativa trata de tudo que resulta da segurança sob o ponto de vista organizacional no seio da empresa, como a seleção do pessoal responsável pela segurança dos sistemas informatizados. Já a política de segurança lógica define as regras de acesso e de circulação das informações no sistema informático. Esta política define os controles para o acesso lógico das informações, especificando “quem” tem acesso a “que” e em “quais”

circunstâncias. Políticas de segurança lógica podem ser decompostas em **políticas de autenticação** e **políticas de autorização** [2]. Quando um usuário é identificado e autenticado, a política de autorização deve especificar quais são as operações que este usuário particular pode realizar no sistema.

As entidades responsáveis e autorizadas pela política de segurança para efetuar acessos às informações mantidas por um sistema são denominadas de **principais**. A granularidade da aplicação da política pode determinar o principal como um usuário, um processo ou ainda uma máquina hospedeira em uma rede. As entidades não autorizadas, chamadas de **intrusos**, podem ser identificadas como usuários, processos, máquinas hospedeiras, etc. (Westphall, 2000).

É importante frisar que uma política de segurança sempre se aplica a um sistema específico, e não a uma classe geral de sistemas. Por sua vez, um **modelo de segurança** é uma representação restrita de uma classe de sistemas que abstrai detalhes de modo a realçar uma propriedade específica ou um conjunto de comportamentos (Obelheiro, 2001). Os modelos de segurança são úteis como guias na definição de políticas específicas de segurança já que são formas de descrever tanto o comportamento de entidades quanto as regras que definem a evolução de uma política de autorização.

Finalmente, os **mecanismos de segurança** em um sistema constituem a implementação das políticas de autorização e de autenticação definidos para o referido sistema. Esses mecanismos asseguram que todos os acessos a objetos no sistema são autorizados pela política definida.

Concluindo, uma política de segurança é um conjunto de regras definidas para atender a um objetivo particular — a segurança computacional — estabelecendo o que é um comportamento aceitável e o que é um comportamento inaceitável por parte dos usuários do sistema. Um sistema sem política de segurança é como uma sociedade sem leis (Landwehr, 2001).

2.2 Mecanismos de Segurança

Os mecanismos de segurança que implementam as políticas de autorização e de autenticação em um sistema distribuído fazem uso de controles de acesso e controles criptográficos (Denning, 1982). Nesta seção primeiramente, serão definidas a autenticação e a autorização, para em seguida apresentar os controles que as implementam.

Em 1985, o Departamento de Defesa dos EUA (*DoD*) definiu o conceito de **Base Computacional de Segurança** ou TCB (*Trust Computing Base*) como o conjunto de mecanismos de proteção necessários para implantação da política de segurança de um sistema, incluindo

[2] Neste documento, de agora em diante, quando o termo política de segurança for utilizado, este refere-se a política de segurança lógica.

o *hardware*, o *software* e o *firmware* (DoD, 1985). O TCB é composto pelo núcleo de segurança, que implementa o conceito de monitor de referência, e também pelos controles adicionais, como os controles criptográficos entre outros.

2.2.1 Autenticação e Autorização

A **autenticação** consiste em um conjunto de procedimentos que permite que uma entidade, também chamada principal (um usuário ou outro sistema), comprove sua identidade perante um sistema. Para isto, a entidade precisa fornecer uma prova de identificação dando evidências de que é realmente quem diz ser. Por exemplo, em muitos sistemas operacionais, são fornecidos aos usuários um identificador (*login*) para que estes possam ser reconhecidos pelo sistema. Para garantir que a identidade de um usuário esteja correta, uma senha (*password*) é associada ao identificador do usuário (Amoroso, 1994). Além da prova de identificação baseada em um segredo que somente o dono da identidade o conheça, como exposto no exemplo acima, outras provas podem ser usadas: um dispositivo que apenas o dono da identidade o possua, como um cartão magnético ou um *smartcard* ou um traço pessoal que seja uma característica única (p. ex.: íris e impressão digital).

Mecanismos de autenticação para sistemas distribuídos tomam a forma mais ampla de um serviço de autenticação. Nestes sistemas, é comum a necessidade de autenticação mútua entre dois principais (para que cada um tenha certeza de com quem está se comunicando) e também de autenticação da origem de dados que permite comprovar a identidade do principal que originou uma determinada mensagem . Geralmente, são os controles criptográficos que implementam estes serviços de autenticação.

A **autorização** é a função que decide se as requisições de acesso a objetos [3] feitas por sujeitos [4] devem ser ou não permitidas. No modelo conceitual introduzido por Lampson (1971), a entidade abstrata responsável pela concretização da autorização, chamada de monitor de referência, é a entidade que recebe todas as requisições e permite ou nega o acesso, de acordo com a política de segurança implantada (ver Figura 2.1).

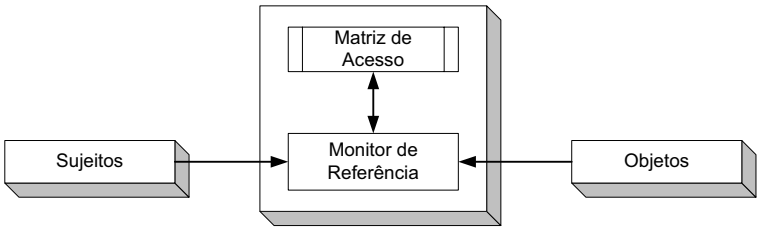


Figura 2.1: Monitor de Referência

[3] Entidades passivas que armazenam informações no sistema (p.ex,: arquivos e segmentos de memória).
[4] Entidades ativas em um sistema computacional que requisitam recursos.

A implementação dos processos de autenticação e de autorização em sistemas distribuídos, bem como as possíveis configurações para estes controles, serão discutidos na seção 2.4.

2.2.2 Controles Criptográficos

Os algoritmos criptográficos[5] podem ser: **algoritmos simétricos**, que usam a mesma chave para cifrar e decifrar mensagens, também conhecidos como algoritmos de chave secreta; ou **algoritmos assimétricos**, que usam uma chave pública para cifrar mensagens e uma chave privada para decifra-las, também conhecidos como algoritmos de chave pública. Como exemplo de algoritmos simétricos ou de chave secreta, pode-se citar: o AES (*Advanced Encryption Standard*) que normalmente usa chaves de 128, 192 ou 256 bits; o RC5 que usa chaves de tamanho definido pelo usuário; e, o IDEA que usa chaves de 128 bits. Entre os algoritmos assimétricos ou de chave pública se destacam o RSA (*Rivest Shamir Adelman*) e o ElGamal.

Os controles criptográficos são usados em sistemas computacionais para prover: confidencialidade às informações, autenticidade de mensagens e de principais; integridade às informações armazenadas ou transmitidas; e não-repudiação. A confidencialidade é garantida através da cifragem de dados e de mensagens. Visando prover a autenticidade de mensagens e a sua integridade, a técnica de **assinatura digital** baseada em chave pública vem sendo amplamente utilizada. Para assinar uma mensagem, um emissor, usando uma função *hash one-way* livre de colisões, deve gerar um resumo criptográfico da mensagem e o resultado deve ser cifrado com a sua chave privada. Esta assinatura deve ser enviada junto com a mensagem. Para verificar a assinatura, é preciso decifrar a assinatura com a chave pública do emissor (comprovando a identidade da origem da mensagem) e recalcular o resumo da mensagem. Quando comparados, se os resumos (o cifrado e o calculado) forem iguais, a assinatura será autêntica (a integridade da mensagem está preservada). Para garantir a não-repudiação, recipientes criptografados são criados para que os autores das mensagens não possam negar o envio dessas mensagens.

Os controles criptográficos são muito importantes para a implementação das políticas de segurança de um sistema, porém o seu uso introduz algumas preocupações quanto ao armazenamento e à distribuição das chaves criptográficas. Técnicas seguras de gerenciamento de chaves precisam ser utilizadas visando amenizar essas preocupações. A forma de distribuição de chaves criptográficas mais empregada atualmente são as que se baseiam em **certificados digitais**, que vinculam chaves públicas a principais. Para ter certeza de que a chave pública contida no certificado realmente pertence ao sujeito do certificado, é preciso que este certificado seja assinado por uma entidade confiável, chamada de autoridade

[5] Algoritmos criptográficos correspondem a uma ou mais transformações matemáticas usadas para cifrar e decifrar informações.

certificadora (AC).

Os processos de autenticação e autorização são baseados no uso de controles criptográficos para geração de certificados de nomes e certificados de autorização. Os certificados de nomes, normalmente, correspondem à informação que serve para assegurar a autenticidade de principais e de objetos do sistema. Já os certificados de autorização servem para garantir a autenticidade de permissões de acesso. O serviço de autenticação, na produção de certificados, pode envolver assinaturas digitais e a distribuição de chaves de sessão.

2.2.3 Controle de Acesso

Os mecanismos de controle de acesso são aqueles usados para implementar a política de autorização de um sistema e estes estão ligados aos modelos de controle de acesso. Os modelos de controle de acesso são comumente classificados em três classes (Sandhu e Samarati, 1996; Obelheiro, 2001):

- **Controle de Acesso Discricionário** (*Discretionary Access Control* - DAC). Os direitos de acesso a cada informação são manipulados livremente pelo responsável da informação (geralmente o proprietário), à sua discrição. Um controle discricionário permite que os dados sejam livremente copiados de objeto para objeto, de modo que mesmo que o acesso aos dados originais seja negado, um sujeito sempre pode obter acesso a uma cópia.
- **Controle de Acesso Obrigatório** (*Mandatory Access Control* - MAC). Baseia-se em uma administração centralizada de segurança, que dita regras incontornáveis que se acrescentam às regras ditas discricionárias. Um controle obrigatório supõe que os usuários e objetos (recursos do sistema) foram etiquetados. As etiquetas dos objetos seguem uma classificação específica, enquanto os usuários ou sujeitos de acesso possuem níveis de habilitação (*clearance*). Os controles que verificam a autorização do acesso são baseados na comparação da habilitação do sujeito e da classificação do objeto.
- **Controle de Acesso Baseado em Papéis** (*Role-Based Access Control* - RBAC). Requer que os direitos de acesso sejam atribuídos a papéis e não a usuários, como no DAC; os usuários obtêm estes direitos em virtude de terem papéis a si atribuídos.

Alguns modelos clássicos correspondentes às classes de controle de acesso são: o Modelo de Matriz de Acesso – segue o DAC; os Modelos Baseados em Reticulados (*Lattices*) – seguem o MAC.

Segundo Landwehr (1981), o uso de modelos de segurança são úteis para o projeto de esquemas de autorização. As políticas de autorização são geralmente descritas tendo como

base estes modelos que permitem verificar se a política está completa e coerente (Nicomette, 1996). Tendo como base um modelo que explicita algumas propriedades que devem ser verificadas na implantação de uma política de segurança, a implementação de um esquema de autorização, seguindo este modelo, fornece aos projetistas argumentos convincentes sobre o funcionamento do sistema.

2.3 Objetivos de Segurança

Dada a especificação de uma política de segurança, os mecanismos de segurança que a implementam podem prevenir ataques, detectar ataques, ou recuperar-se de ataques (Bishop, 2003). Esses objetivos de segurança podem ser adotados separadamente ou de forma combinada.

Mecanismos orientados a **prevenção** são projetados para evitar que ataques provoquem as violações de segurança. Em sistemas militares, a primeira geração de medidas de segurança computacional tinha como objetivo a prevenção de violações, em especial, a revelação de dados sensíveis (Landwehr, 2001). Prevenção envolve, tipicamente, a implementação de mecanismos nos quais usuários não autorizados não podem ultrapassá-los. Esses mecanismos devem ser implementados de forma inalterável, correta e confiável (Bishop, 2003).

A **Detecção** é útil principalmente quando ataques não podem ser prevenidos. Mecanismos de detecção admitem que ataques possam ocorrer; o objetivo então é determinar quando um ataque está em progresso ou foi concluído com sucesso e reportá-lo aos controles do sistema (Bishop, 2003). A segunda geração de tecnologias de segurança, caracterizada por *firewalls* e sistemas de detecção de intrusão, tem por objetivo detectar e limitar, pelo menos, as violações de segurança que não podem ser prevenidas (Landwehr, 2001). Os recursos protegidos por mecanismos de detecção são continuamente ou periodicamente monitorados.

Segundo Bishop (2003), a **recuperação** tem duas formas. A primeira é interromper o ataque, avaliar e reparar qualquer dano causado pelo mesmo. Por exemplo, se um invasor remover um arquivo, um mecanismo de recuperação deve restaurar o arquivo a partir das fitas de *backup*. Por definição, a recuperação requer o reinício da operação correta do sistema. Na segunda forma de recuperação, o sistema continua funcionando corretamente enquanto um ataque estiver em execução. Estes mecanismos estão baseados em técnicas de tolerância a faltas bem como em técnicas de segurança e são, tipicamente, usados em sistemas críticos de segurança (*safety*). Este difere da primeira forma de recuperação porque em nenhum ponto o sistema deve funcionar incorretamente. Todavia, o sistema pode desabilitar funcionalidades não essenciais. Segundo Landwehr (2001), a terceira geração de tecnologias e arquiteturas de segurança tem que ter a habilidade para tolerar ataques e continuar suas funções críticas, apesar de operar com modo degradado, enquanto um ataque está em execução.

2.4 Mecanismos de Autenticação e de Autorização em Sistemas Distribuídos

A implementação de mecanismos de segurança em sistemas distribuídos, que envolvem vários domínios administrativos e tecnologias heterogêneas, é uma tarefa bastante complexa. A escala do sistema aumentando, a implantação dos mecanismos de autenticação e de autorização nos sistemas distribuídos, torna-se ainda mais complexa. Um sistema é dito escalável se este puder tratar a adição de usuários e recursos sem sofrer uma perda notável de desempenho ou um aumento da complexidade de administração (Neuman, 1994). Nestes sistemas, a autenticação está principalmente ligada à verificação das identidades de principais e a autorização define os acessos permitidos aos recursos e informações nestes sistemas. A questão que se coloca é como implementar estes serviços em sistemas distribuídos de larga escala permitindo que estes possam evoluir durante a sua dinâmica.

Como já comentado, há uma dependência natural entre os mecanismos de autenticação e autorização. As configurações mais comuns das abordagens de implementação destes serviços, encontradas na literatura e em alguns estudos de caso, serão também apresentados, a seguir.

2.4.1 Abordagem Centralizada

Nesta abordagem, a gestão do serviço de autenticação e autorização pode ser implementada de forma centralizada por uma única máquina no sistema distribuído. Esta abordagem pode ter a vantagem de permitir uma política de autorização coerente e fácil de se implantar; porém, a centralização dos controles provocaria um baixo desempenho e um gargalo no acesso aos serviços de autenticação e autorização. Além disso, a concentração de funções em uma só máquina cria um ponto vulnerável a falhas capaz de comprometer a segurança de todo o sistema. Portanto, esta abordagem se apresenta inviável para sistemas distribuídos.

2.4.2 Abordagem de Autenticação Centralizada e Autorização Descentralizada

O seccionamento das funções de controle da segurança pode levar a uma abordagem com a autenticação centralizada e a autorização descentralizada. Neste caso, a autenticação, em um domínio de nomes, é conduzida por um modelo centralizado, devido ao seu vínculo com o espaço de nomes, enquanto a autorização é controlada independentemente por cada sítio do sistema. A gestão descentralizada da política de autorização do sistema através da distribuição de controle de direitos de acesso (vários servidores decidindo concorrentemente) pode provocar alguns problemas de coerência, já que neste caso a matriz de acesso se encontra seccionada entre os sítios da rede. O problema da confiança no servidor de autenticação (ponto único de falhas do sistema) continua nesta abordagem.

Este modelo é adequado em redes corporativas. Quando o ambiente é um sistema de larga escala, como a Internet, soluções para resolução de nomes se fazem necessárias. A escalabilidade é alcançada nesta abordagem através de modelos de hierarquias de domínios de nomes.

O projeto Athenas, desenvolvido no MIT, segue essa distribuição de funções utilizando o Kerberos (Steiner *et al.*, 1988; IEFT, 1999) como servidor único de autenticação. O *framework* X.509 (ITU-T, 1993) e o sistema Delta-4 (Powell, 1988) também estão baseados nesta abordagem.

2.4.3 Abordagem Descentralizada

Em uma abordagem totalmente descentralizada, cada sítio do sistema possui uma Base Computacional de Segurança (TCB), responsável por controlar os acessos locais e remotos. A forma mais comum de se implementar esta abordagem descentralizada é através do uso de redes de confiança. Em uma rede de confiança, as entidades consideradas confiáveis são distribuídas pela rede que se encarregam da implementação das políticas de autenticação e autorização. Nesta abordagem, o problema da gestão descentralizada continua comprometendo a coerência da política de autorização. O TCSEC (*Trusted Computer System Evaluation Criteria*)(DoD, 1985) e o SPKI/SDSI (*Simple Public Key Infrastructure - Simple Distributed Security Infrastructure*) (Elisson, 1999; Lampson e Rivest, 1996) seguem esta abordagem.

2.5 Estudo de Casos

2.5.1 Kerberos

O Kerberos, desenvolvido no MIT em 1980, foi projetado para fornecer autenticação entre clientes e servidores em redes que formam um domínio de gerenciamento único (Intranets)(Neuman e TS'o, 1994). Como tecnologia de segurança, o Kerberos vem sofrendo diversas revisões e aprimoramentos; a versão 5 é um padrão IETF(*Internet Engineering Task Force*)(IEFT, 1999) e vem sendo usada por diversas empresas e universidades. O Kerberos está enquadrado na abordagem de **controle de autenticação centralizada e autorização descentralizada**. Os clientes que desejarem acessar informações em servidores de aplicação devem, obrigatoriamente, se fazer autenticar pelo servidor único de autenticação Kerberos (ver Figura 2.2). Este cliente obterá uma prova de sua autenticidade para apresentar aos servidores de aplicação que implementam os mecanismos de autorização. O servidor Kerberos deve ser considerado confiável tanto pelos clientes quanto pelos servidores de aplicação. Esta relação de confiança está baseada na posse de uma chave secreta compartilhada entre um cliente, pertencente a um domínio Kerberos e o servidor Kerberos.

Os serviços de autenticação Kerberos para ser escalável faz uso de domínios separados de autenticação, chamados de *realms* (cada domínio tem o seu próprio servidor Kerberos) (IEFT, 1999). Para que um cliente de um domínio Kerberos possa ter acesso a serviços em um outro domínio Kerberos, é necessário que os servidores de autenticação destes domínios possuam uma relação de confiança entre si (compartilhem uma chave secreta). Como no Kerberos, o sistema de nomes concatena os nomes locais aos nomes do domínio, é possível ter nomes únicos em um sistema de larga escala (IEFT, 1999).

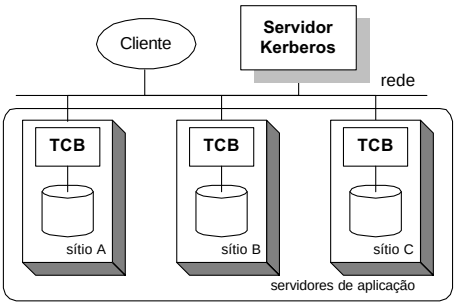


Figura 2.2: Modelo Kerberos

As desvantagens desta abordagem já foram comentadas na seção 2.4.2, porém vale acrescentar que a dependência do compartilhamento de chaves secretas com cada cliente pertencente ao domínio com o servidor Kerberos também traz outra desvantagem para esta abordagem. Mais detalhes sobre a dinâmica do processo de autenticação do Kerberos pode ser encontrada em Couloris e Dollimore (2001); Steiner *et al.* (1988); IEFT (1999).

2.5.2 Modelo X.509

No modelo X.509 (ITU-T, 1993), padrão ISO (*International Organization for Standardization*), o serviço de autenticação também é centralizado, porém o uso de nomes X.500 aumenta a aplicação do modelo em redes de larga escala como a Internet (Garfinkel e Spafford, 1997). O X.500, através de uma hierarquia de nomes, permite uma nomeação global e geograficamente distribuída.

Neste modelo, assim como no modelo Kerberos, **a autorização é descentralizada**. Após a autenticação, o cliente fica sujeito ao controle de acesso do servidor de aplicação. O X.509 está baseado em uma infra-estrutura de chave pública ou ICP [6], em que um cliente particular possui um par de chaves: uma chave privada que fica em seu poder, e outra, chave pública, que fica armazenada em diretórios X.500 na forma de certificados emitidos pela autoridade certificadora (AC) do domínio[7]. A adoção de uma ICP para a gerência das chaves se enquadra ainda na abordagem de **controle centralizado de autenticação**, pois o servidor de

[6] Uma ICP contém os serviços necessários para desenvolver e suportar tecnologias baseadas em chave pública. A infra-estrutura do X.509 trata de questões relacionadas às autoridades certificadoras, ao formato dos certificados, à revogação de certificados e às políticas de segurança nas quais uma chave pública torna-se confiável.

[7] Segundo a recomendação X.509, versão 3, as ACs geram certificados de nomes e de autorização.

aplicação não intermedia o processo de autenticação, mas apenas participa do mesmo quando necessário, produzindo certificados que associam uma chave pública a um determinado principal do domínio.

As comunidades X.509 devem ser construídas com base na confiança das chaves privadas das ACs, que estão organizadas em uma hierarquia global. Uma AC controla sua chave privada, que é usada para assinar os certificados dos quais a AC é a emissora. Estes certificados vinculam nomes de validade global a chaves públicas. O modelo baseia-se principalmente em certificados formando **cadeias de autenticação** que partem de uma chave pública de uma AC confiável até uma chave pública de um usuário. Este modelo de certificação é usado em diversos controles criptográficos, como por exemplo, o SSL/TLS que suporta autenticação usando chaves públicas e certificados X.509.

Na infra-estrutura X509, é possível construir relações de confiança através de certificação cruzada entre duas ACs, objetivando dispensar a necessidade de percorrer toda a estrutura hierárquica para ir de uma AC até outra (NIST, 2000). Porém, em tal modelo, este tipo de alternativa de fluxo só é permitido entre autoridades certificadoras, o que não descaracteriza a AC como entidade centralizadora da certificação nem a rigidez e complexidade da estrutura hierárquica imposta por esta infra-estrutura. A AC e o diretório X.500, que armazena os certificados, não necessariamente participam, intermediando o processo de autenticação, como no servidor de autenticação Kerberos. O modelo X.509 possibilita alternativas para o processo de autenticação visando diminuir a dependência de uma entidade única centralizadora no domínio. Um servidor de aplicação pode acessar o serviço de diretório em busca de certificados de principais e, de posse dos certificados, estes podem ser armazenados para usos posteriores, sendo que o servidor precisará somente fazer consultas freqüentes a LCR (Lista de Certificados Revogados). Além disso, as LCRs podem ser publicadas de forma a evitar o gargalo de pontos de acesso únicos. Apesar desses esforços para minimizar a dependência de uma entidade única no armazenamento e na dinâmica da autenticação e, mesmo que a operação da AC seja *offline* no sentido de torná-la menos vulnerável, uma falha ou uma operação não autorizada em uma AC continuará comprometendo a segurança da informação no sistema (Santin, 2004).

O modelo hierárquico de confiança em que se baseiam os certificados X.509 e a tentativa de utilizar um nome que seja único e que identifique globalmente o proprietário do certificado, na prática, não funcionam como inicialmente proposto. Muitas vezes as ACs trabalham de forma individualizada, não havendo nenhuma relação de confiança entre elas. Quando cada AC trabalha de forma individualizada, cada uma com os seus próprios critérios, não há como estabelecer relações de confiança globais.

2.5.3 Modelo TCSEC

O TCSEC (*Trusted Computer System Evaluation Criteria*)(DoD, 1985) segue a **abordagem descentralizada** do controle da autenticação e da autorização. Cada sítio do sistema possui uma TCB responsável por controlar os acessos locais e remotos. Cada TCB realiza as funções de autenticação de seus sujeitos. O conjunto de TCBs compõe a NTCB (*Network Trust Computing Base*). Quando um sujeito deseja acessar um objeto remoto, este deve encaminhar sua requisição ao TCB local que contacta o TCB remoto para autorizar ou negar o acesso ao objeto sob o seu controle (ver Figura 2.3). A segurança global do sistema depende da rede de confiança (NTCB) formada pelos TCBs.

A NTCB é responsável pela autenticação e implementa uma matriz de acesso particionada, envolvendo todos os sítios da rede que controla todos os objetos da NTCB. Devido a isto, os problemas de coerência da política de autorização, discutidos na seção 2.4.3, continuam nesta abordagem. Se uma partição da matriz de acesso for corrompida, o sistema global estará corrompido já que no TCSEC não são descritos mecanismos que permitam detectar elementos corrompidos na rede de confiança.

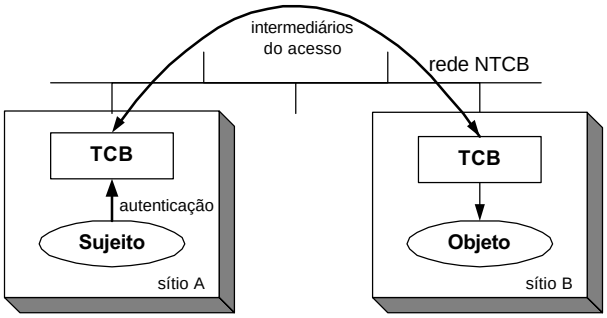


Figura 2.3: Modelo TCSEC

2.5.4 Modelo SPKI/SDSI

O SPKI/SDSI (*Simple Public Key Infrastructure - Simple Distributed Security Infrastructure*), assim como o TCSEC, segue a **abordagem descentralizada**. Motivadas pela complexidade do esquema global de nomeação do X.509, duas iniciativas relacionadas, porém independentes, surgiram: o SPKI, padrão IETF, proposto por Carl Ellison e outros membros (1999), e o SDSI, projetado no MIT por Ronald Rivest e Butler Lampson (1996). Como as iniciativas são complementares estas convergiram para uma só especificação, o SPKI/SDSI, citadas por alguns autores como apenas SPKI. A especificação SPKI/SDSI[8] define uma infraestrutura de chaves públicas e em espaços de nomes locais.

Em Blaze *et al.* (1996), o conceito de gerência de confiança (*trust management*) é introduzido como sendo um conjunto de mecanismos unificados para especificar e validar tanto

[8]No contexto desta tese, será sempre considerada a versão 2.0 da especifi cação SPKI/SDSI.

políticas de segurança como certificados. Estes mecanismos de gerência de confiança são muitas vezes chamados de PKIs, orientados a chave. O SPKI é um exemplo de sistema baseado em gerência de confiança.

O SPKI/SDSI tem um modelo igualitário. Os sujeitos (ou principais) são chaves públicas e cada chave pública é também uma entidade certificadora (Clarke, 2001). Não há infra-estrutura global hierárquica como no X.509. Não há uma entidade centralizadora para o registro de chaves públicas e emissão de certificados como a autoridade certificadora do X.509.

Há dois tipos de certificados no SPKI/SDSI (Clarke, 2001): certificados de nomes e certificados de autorização. Um certificado de nome define um nome local no espaço de nomes locais do emissor e ligam esse nome a uma chave pública ou, ainda, a outro nome [9]. O emissor do certificado é identificado por sua chave pública. Quando certificados de nomes fazem referência a nomes definidos em outros certificados, uma **cadeia de certificados de nomes** é formada. A divulgação de nomes no SPKI/SDSI é feita através da resolução de cadeias de certificação de nomes.

O certificado de autorização concede autorizações específicas do emissor do certificado (administrador do serviço) para o sujeito do certificado e essas autorizações são ligadas a um nome, a um grupo de nomes ou a uma chave. O emissor de um certificado pode ainda permitir que o principal delegue as permissões recebidas a outros principais. A delegação permite uma distribuição controlada da autorização. Um certificado de autorização SPKI deve possuir os seguintes campos: a chave do emissor, o sujeito do certificado, o *bit* de delegação (*boolean*), uma *tag* que especifica uma autorização e uma validade para o certificado.

Quando um sujeito recebe um certificado que permite a delegação, este pode propagar a autorização expressa no certificado. Para isto, este deve anexar o certificado a uma cadeia e enviá-la ao sujeito deste novo certificado, pois esta cadeia será solicitada ao sujeito quando o mesmo for tentar utilizar um recurso protegido. Ao permitir a delegação do certificado, o emissor assume que o sujeito é de extrema confiança, pois o mesmo poderá delegar o certificado a quem ele queira sem que necessite a autorização prévia do emissor. Assim, este modelo de delegação possibilita a construção de cadeias de confiança que partem do administrador de um serviço terminando em chaves de principais clientes deste serviço. Quando um determinado sujeito ou chave pública deseja obter acesso a algum recurso, este deve apresentar uma requisição assinada e uma cadeia de certificados que permita a checagem das autorizações concedidas (Clarke, 2001). Um dos principais objetivos da checagem das cadeias de certificados é assegurar que dentro da cadeia de certificados, mesmo que o emissor no topo da cadeia conceda o direito de delegação subseqüentes nenhum sujeito pode obter autorizações maiores que as do próprio emissor de origem (Elien, 1998).

[9] Estes certifi cados estão baseados em nomes SDSI (Lampson e Rivest, 1996) que induz ao uso de nomes locais, mesmo no sentido global de sistemas distribuídos.

O SPKI/SDSI tem avançado bastante no sentido de se firmar como uma plataforma de suporte à autenticação e à autorização em sistemas distribuídos. A abordagem totalmente descentralizada para o controle da autorização e autenticação seguida pelo SPKI/SDSI se mostra mais adequada para a implementação desses controles de segurança em sistemas distribuídos de larga escala. A dificuldade deste modelo está em identificar, entre os certificados de um cliente, caminhos de uma cadeia de confiança que levem ao servidor desejado. Em alguns casos, um cliente e um servidor podem não estar conectados por uma cadeia de confiança. Este problema é de difícil solução, porém diversos trabalhos se dedicam à determinação de cadeias de certificados que viabilizem acessos (Santin, 2004).

2.5.5 Comparação entre as Configurações de Controle da Autenticação e Autorização

As características e limitações das possíveis configurações de controle de autenticação e de autorização em sistemas distribuídos já foram analisadas na seção 2.4. A Tabela 2.1 apresenta um resumo dos principais aspectos para implantação da autorização e da autenticação, considerando os quatro mecanismos apresentados nas seções anteriores.

Mecanismo	**Autenticação**	**Autorização**	**Espaço de nomes**	**Escalabilidade**	**Interoperabilidade**	**Tolerân. a falhas**
Kerberos	centralizada	descentralizada	local	sim	sim	-
X.509	centralizada	descentralizada	global	sim	sim	-
TCSEC	descentralizada	descentralizada	local	limitada	-	-
SPKI/SDSI	descentralizada	descentralizada	local	sim	sim	sim

Tabela 2.1: Comparação dos Mecanismos de Autenticação e Autorização

As principais limitações da abordagem de autenticação centralizada e da autorização descentralizada, seguidas pelo Kerberos e pelo X.509, são a existência de um ponto único de falhas e de vulnerabilidades (entidade centralizadora) e o comprometimento do desempenho. Como analisado anteriormente, tanto o Kerberos quanto o X.509 utilizam algum tipo de mecanismo ou técnica visando minimizar os efeitos da centralização da autenticação. O X.509, que adota o sistema de chave pública e o serviço de diretório X.500, tenta suavizar esses efeitos da centralização através da replicação do serviço de diretório. Já o Kerberos se apresenta como o mais limitado devido às restrições impostas pelo uso de chave secreta.

A abordagem, totalmente descentralizada, seguida pelo TCSEC e pelo SPKI/SDSI, se mostra mais adequada para o controle da autenticação e da autorização em sistemas distribuídos de larga escala. Na proposta TCSEC, os problemas de coerência da política de autorização não são tratados adequadamente e alguns aspectos das interações nas redes de confiança não são detalhados, limitando assim a sua aceitação. De maneira geral, o SPKI/SDSI se apresenta como uma poderosa solução para implementação de mecanismos de autenticação e de autorização, porém, a necessidade do prévio conhecimento das correntes de certificados de autorização se mostra, em certas circunstâncias, como um problema de difícil solução.

2.6 Conclusões do Capítulo

A medida em que a infra-estrutura da tecnologia da informação se torna mais extensa e complexa, aumenta o desafio de integrar com segurança a tecnologia com os diversos ambientes de aplicação (p.ex.: sistemas bancários, comércio eletrônico e redes privadas). Políticas de segurança para proteger sistemas distribuídos devem ser adotadas e projetadas para assegurar níveis de segurança apropriados para as atividades que são executadas no sistema; e, mecanismos de segurança devem ser empregados para implementar estas políticas de segurança. Quando baseados em modelos formais de segurança, as políticas de segurança podem ser mais facilmente verificadas quanto à sua completude e coerência.

Quando as aplicações de larga escala são consideradas, a autorização e a autenticação devem evoluir tomando como base modelos onde as relações de confiança possam ser estabelecidas de maneira distribuída, o que favorece soluções escaláveis e flexíveis. O SPKI/SDSI segue a abordagem descentralizada de implementação de mecanismos de autenticação e autorização e encontra escalabilidade sem comprometer a segurança, usando para isto chaves públicas e uma arquitetura de espaço local de nomes. Devido às suas vantagens, a abordagem descentralizada do modelo do SPKI/SDSI foi adotada neste trabalho.

No próximo capítulo, o paradigma de agentes móveis será apresentado e as ameaças de segurança, considerando este paradigma, serão descritas, assim como os mecanismos de segurança que visam contornar essas ameaças também serão analisados.

Capítulo 3

Agentes Móveis

3.1 Introdução

Este capítulo define mobilidade de código, apresenta os problemas discutidos na literatura em relação aos paradigmas baseados em mobilidade e um arcabouço conceitual, tendo em vista as tecnologias de códigos móveis, os estilos de arquiteturas e os domínios de aplicações definidos para modelos de códigos móveis. O paradigma de agentes móveis será apresentado elucidando os principais conceitos relacionados a este paradigma e suas vantagens e desvantagens serão analisadas.

A habilidade de um agente móvel, ao se transportar através de uma rede de larga escala, permite o desenvolvimento de serviços e aplicações mais flexíveis e dinâmicos se comparados aos modelos clássicos como o cliente-servidor. Apesar disso, a ampla aceitação dos agentes móveis vem sendo prejudicada devido às questões de segurança e confiabilidade.

3.2 Mobilidade de Código

O conceito de mobilidade de código não é recente. Na década de 60, sistemas de entrada de *jobs* remotos já eram usados para submeter programas a um computador central. Entretanto, mais recentemente, a mobilidade de código se tornou popular através do uso dos *applets* Java em navegadores Web (Jansen, 2000).

Diferentes significados são encontrados na literatura para o termo **código móvel**. A falta de uma base conceitual sedimentada complica, por exemplo, a atividade de selecionar a melhor tecnologia de código móvel para uma dada aplicação, assim como dificulta a estimativa dos benefícios que essa tecnologia pode trazer já desde a fase de projeto da aplicação (Picco, 1998). Segundo Thorn (1997), um código móvel em uma rede de larga escala pode ser visto como um software que viaja através de uma rede heterogênea, atravessando diversos

domínios de segurança e sendo executado automaticamente no seu destino. Domínios de segurança podem ser grandes, como uma rede corporativa de computadores ou pequenos como um computador de bolso (*hand-held digital assistant*).

A idéia chave da mobilidade de código contribui para fornecer uma solução complementar para a estrutura de aplicações distribuídas do tipo cliente-servidor tradicional (CS), baseadas em RPC (*Remote Procedure Call*). Novas linguagens de programação, fornecendo suporte para alguma forma de mobilidade de código, foram propostas nos últimos cinco anos. Estas linguagens são usualmente conhecidas como linguagens de código móvel ou MCLs (*mobile code languages*) que, entre outras coisas, apresentam a capacidade para reconfigurar dinamicamente, em tempo de execução, a ligação entre os componentes de software da aplicação e a sua localização física, dentro de uma rede de computadores (Carzaniga *et al.*, 1997).

Será adotada neste trabalho uma terminologia proposta por Picco (1998) que visa fornecer meios para classificar e comparar abordagens existentes no contexto de códigos móveis. O autor apresenta um arcabouço conceitual para mobilidade de código tendo em vista três dimensões, a saber:

- **Tecnologias de códigos móveis**. São as linguagens e sistemas que fornecem mecanismos habilitando e suportando mobilidade de código. Essas tecnologias são usadas pelo desenvolvedor de aplicações no estágio de implementação;
- **Estilos de Arquitetura**. São estilos de arquiteturas que os desenvolvedores de aplicações usam na definição da aplicação. Um estilo de arquitetura identifica uma configuração específica para os componentes e suas interações;
- **Domínios de Aplicação**. São classes de aplicação que compartilham o mesmo objetivo geral; por exemplo, aplicações de comércio eletrônico.

Nas seções subseqüentes, serão tratados os conceitos relacionados a cada um dos tópicos apresentados acima.

3.2.1 Tecnologias de Códigos Móveis

A Figura 3.1 compara a estrutura dos sistemas distribuídos tradicionais com a dos sistemas de códigos móveis. Até a camada do Sistema Operacional de Rede a arquitetura de ambos os sistemas é idêntica. Na última camada dos sistemas distribuídos tradicionais, é oferecida uma plataforma onde componentes localizados em máquinas distintas são vistos e utilizados pelo usuário como se estivessem na sua própria máquina. Através dessa transparência, não é necessário o conhecimento das características da rede onde o sistema é executado. Os serviços e facilidades do CORBA (OMG, 2004) são exemplos desta abordagem.

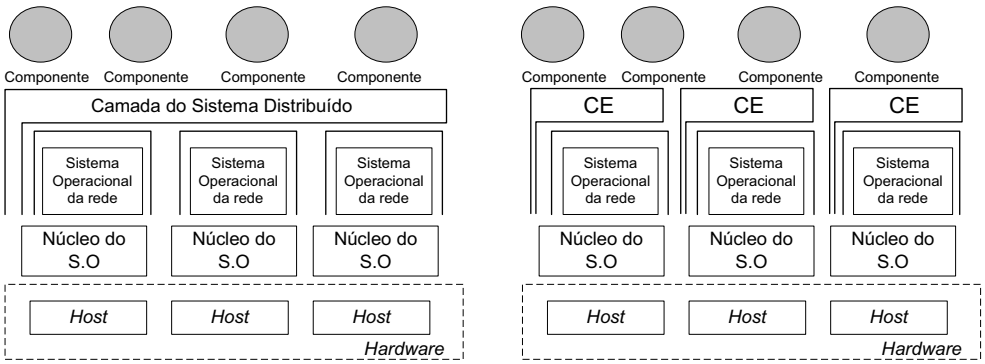


Figura 3.1: Sistemas Tradicionais x Sistemas de Códigos Móveis

Nos sistemas de códigos móveis, a camada do sistema distribuído, que fornece a transparência da rede para os sistemas tradicionais, é substituída pelos Ambientes Computacionais (*Computational Environments* ou CEs) que retém a "identidade"do sítio onde este está localizado. O propósito do ambiente computacional é dar suporte às aplicações com a capacidade de realocar dinamicamente seus componentes sobre diferentes sítios ou endereços de rede. A transparência de rede não é necessária nestes sistemas.

Os ambientes computacionais são formados pelas unidades de execução (*executing units* ou EUs) e pelos recursos que podem ser compartilhados entre múltiplas unidades de execução. Unidades de execução representam fluxos seqüenciais de computação. Exemplos típicos de EU são os processos com uma única *thread* ou *threads* individuais de um processo *multi-thread*.

Com relação à forma como ocorre a migração da unidade de execução, os sistemas de códigos móveis oferecem duas formas de mobilidade (Picco, 1998):

- **mobilidade forte** (*strong mobile*): é a habilidade que permite a migração do código e de todo o estado de execução de uma unidade de execução para um ambiente computacional diferente;
- **mobilidade fraca** (*weak mobile*): é a habilidade que permite transferência de código através de diferentes ambientes computacionais; o código pode ser acompanhado por algum dado de inicialização, mas não há migração de todo o estado de execução.

Mobilidade forte é suportada por duas técnicas: **migração** e **clone remoto**. A técnica de migração compreende em suspender uma unidade de execução, transmiti-la para o ambiente computacional destino e então executá-la novamente. A técnica de clone remoto envolve a criação de uma cópia de uma unidade de execução em um outro ambiente computacional. Esta difere da técnica da migração porque após a clonagem da unidade de execução para um novo ambiente computacional, a unidade de execução original não é gerenciada pelo ambiente computacional onde a unidade clonada se encontrava.

Técnicas, suportando mobilidade fraca, fornecem a capacidade para transferir o código através de ambientes computacionais e ainda ligá-los dinamicamente com a unidade de execução ativa, ou usá-los como segmento de código para uma nova unidade de execução. Cada técnica pode ser classificada de acordo com a direção da transferência do código, a natureza como o código se move, a sincronização envolvida, e até mesmo quando o código será executado no sítio destino.

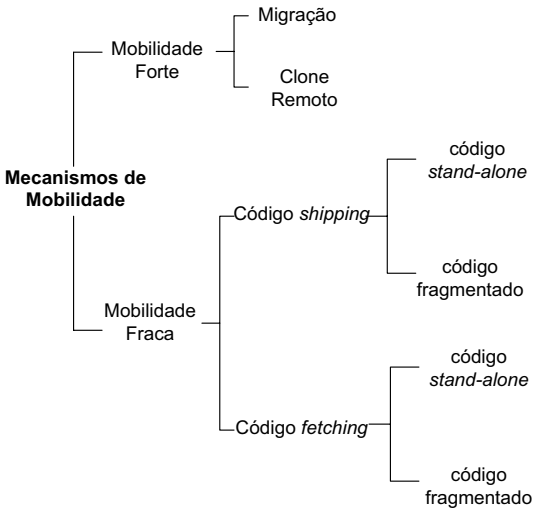


Figura 3.2: Classificação das Técnicas de Mobilidade

Com relação à direção da transferência do código, uma unidade de execução pode buscar (*fetch*) o código para ser dinamicamente ligado e/ou executado, ou pode ainda enviar (*ship*) o código para outro ambiente computacional. O código migrado pode ser *stand-alone* ou fragmentado. O código *stand-alone* é autocontido e será usado para instanciar uma nova unidade de execução sobre o sítio destino. Já o código fragmentado deve ser ligado a um contexto de um código que já está em execução e eventualmente será executado. A Figura 3.2 apresenta a classificação geral dos mecanismos de mobilidade descrita acima.

As linguagens de códigos móveis diferem na forma como estas suportam mobilidade de acordo com a classificação apresentada anteriormente. A tabela 3.1 compara o suporte à mobilidade de algumas MCLs.

3.2.2 Estilos de Arquiteturas ou Paradigmas de Projeto

Arquiteturas de *softwares* com características similares podem ser representadas por estilos de arquiteturas ou paradigmas de projeto que definem abstrações e estruturas de referência que podem ser instanciadas dentro de um projeto de um sistema. Abordagens tradicionais para o projeto de *software* não são suficientes quando aplicações de larga escala, que exploram mobilidade de código e reconfiguração dinâmica de componentes de *software*, precisam ser projetadas. Neste caso, os conceitos de localização, a distribuição de componentes e a

MCLs	Desenvolvedores	Mobilidade	Mecanismos	Comentários
AgentTcl	Dartmouth College (Estados Unidos)	Forte	Migração, Clone Remoto e *Shipping*	Passou a se chamar D'agents
Ara	Univ. de Kaiserlautern (Alemanha)	Forte	Migração	Suporta C, C++ e Tcl
Java	Sun Microsystems	Fraca	Code Fetching	Applets são aplicações típicas de cód. móveis
Java **Aglets**	IBM (Tóquio)	Fraca	*Code Fetching* e *Code Shipping*	Aglets = agent + applets
TACOMA	Troms And COrnell Mobile Agents	Fraca	*Code Shipping*	Tcl estendida
Telescrip	General Magic	Forte	Migração e Clone Remoto	Primeira MCL

Tabela 3.1: Tabela Comparativa de MCLs

migração de componentes para diferentes localizações precisam ser explicitamente levados em conta ainda durante a fase de projeto.

Antes de introduzir os estilos de arquiteturas, alguns conceitos básicos de entidades que constituem um sistema de *software* serão elucidados. Componentes são os constituintes de uma arquitetura de *software*. Estes podem ainda ser divididos em: **componentes de código**, que encapsulam o conhecimento ou regras para executar uma computação particular (*know-how*); **componentes de recursos**, que representam dados ou dispositivos usados durante a computação; e **componentes computacionais ou de execução**, que são executores ativos capazes de processar um componente de código e gerenciar componentes de recursos, conforme especificado em um conhecimento específico. As **interações** são eventos de comunicação envolvendo dois ou mais componentes, como por exemplo, uma mensagem trocada entre dois componentes computacionais. O conceito de **sítios**, intuitivamente, implica a noção de local. Os sítios hospedam componentes, além de suportar componentes computacionais, exigindo recursos de processamento.

As arquiteturas descritas, a seguir, são analisadas em termos das interações ocorridas entre componentes, considerando a coordenação e localização destes, ao prestar um serviço qualquer (Picco, 1998). O cenário é montado de forma que um componente computacional A, localizado no sítio S_A, requisita a execução de um serviço qualquer. Em se tratando de um sistema distribuído, assume-se também a existência de um outro sítio S_B, com componente computacional B, necessário na realização do serviço. As arquiteturas são analisadas levando-se em conta o local dos componentes antes e depois da prestação do serviço, o componente computacional responsável pela execução do serviço e, finalmente, o local onde a computação é efetuada. A Figura 3.3 resume, com base nestes fatores, as quatro arquiteturas apresentadas em detalhes, de acordo com (Picco, 1998; Papaioannou, 2000).

- **Cliente-Servidor** (C/S). Na arquitetura cliente-servidor um componente computacional B, localizado no sítio S_B, recebe requisições para prestação de alguns serviços,

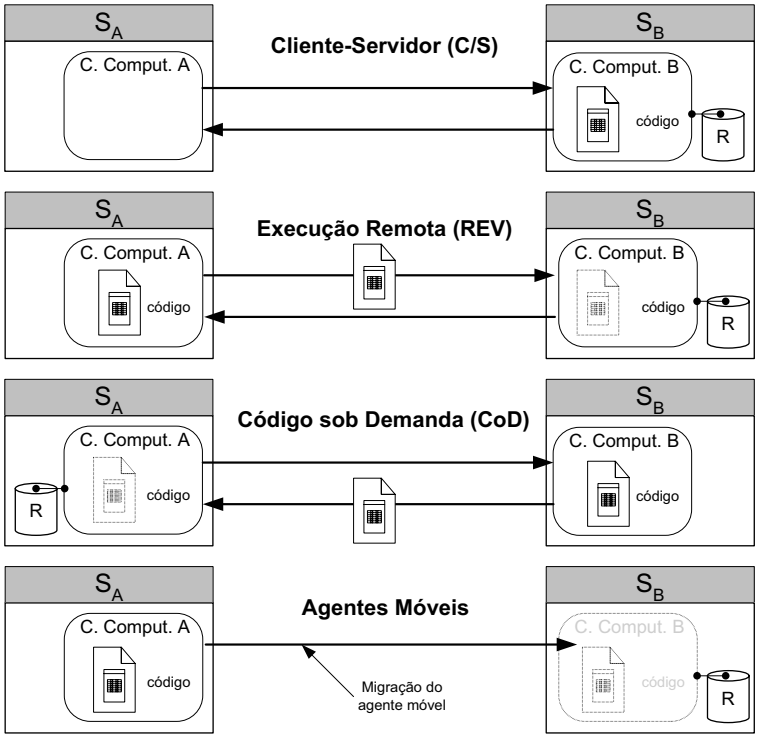


Figura 3.3: Paradigma de Códigos Móveis

sendo chamado de servidor. O componente de código para prestação do serviço, assim como os recursos utilizados para tal, estão localizados no sítio S_B. Um componente A, chamado de cliente, localizado em um sítio S_A, requisita a execução de um serviço oferecido pelo servidor, interagindo com o componente B. Ao receber esta requisição, B executa o código referente à prestação do serviço, utilizando seus próprios componentes recursos. Normalmente, a realização de um serviço fornece algum tipo de resultado que é enviado para o cliente que fez a requisição.

- **Execução Remota ou** *Remote Evaluation* **(REV)**. No paradigma REV, um componente A, localizado no sítio S_A, detém o código para prestação de serviço, porém necessita de recursos para a sua execução. Os recursos necessários estão localizados no sítio S_B. Então, A envia o código para B para que este execute o código usufruindo dos próprios recursos localizados no sítio S_B (*shipping*). Os resultados da prestação do serviço são enviados de volta para A.

- **Código sob Demanda (CoD)**. Nesta arquitetura, o componente A possui todos os recursos necessários para a prestação de serviço, faltando-lhe, porém, as instruções de como os recursos devem ser usados para que o serviço possa ser realizado. Desta forma, A interage com o componente B, localizado no sítio S_B, requisitando o código para tal serviço (*fetching*). Ao receber o código de B, o componente A executa-o, utilizando seus próprios recursos.

- **Agentes Móveis**. O paradigma de agentes móveis difere de todos os outros paradigmas citados anteriormente, pois a prestação de um serviço nem sempre envolve uma

interação entre dois componentes. Nesta arquitetura, o componente A, localizado inicialmente no sítio S_A, detém o código e os recursos necessários para a execução do serviço. O serviço pode ainda necessitar de recursos extras localizados em um sítio remoto, S_B. Neste caso, o componente de código migra para o sítio S_B, transferindo consigo o código, o seu estado e recursos móveis como resultados intermediários. Ao chegar no sítio S_B, A completa o serviço usufruindo dos recursos disponíveis neste sítio. Assim, enquanto no REV e no CoD o foco é na transferência de código entre componentes, no paradigma de agentes móveis, o elemento a ser transferido é todo o componente computacional, composto do seu código, estado e alguns recursos necessários para a realização da tarefa.

Picco (1998) constata que os estilos de arquiteturas, apresentados acima, são independentes de uma linguagem ou de sistemas particulares nos quais estes são implementados. Os paradigmas de códigos móveis tratam explicitamente do conceito de localização. A abstração de sítio é introduzida em nível de projeto para que a localização de componentes computacionais seja levada em conta.

O paradigma de código móvel tem-se mostrado eficiente no desenvolvimento de sistemas distribuídos (Picco, 1998). A maioria dos paradigmas tradicionais são estáticos em relação à estrutura do código de seus componentes e do local onde estes são executados. Além disso, os tipos de interações são fixos, uma vez que é impossível mover os componentes a fim de otimizar as interações em tempo de execução. O paradigma de código móvel é caracterizado pela transitoriedade dos componentes nas localidades de um sistema. As arquiteturas REV e CoD, por exemplo, permitem a execução de código em processadores remotos, tornando muitas vezes a troca de dados entre os componentes envolvidos na computação, em simples interações locais. O CoD ainda oferece uma certa flexibilidade aos componentes ao permitir que eles busquem códigos em outros sítios, estendendo seu comportamento.

3.2.3 Domínios de Aplicação de Códigos Móveis

Como o paradigma de código móvel ainda é restrito a um pequeno número de pesquisadores, o domínio de suas ferramentas é incipiente, principalmente quando comparado ao domínio das aplicações baseadas no sistema cliente/servidor tradicional. Isto é uma conseqüência da imaturidade da tecnologia, principalmente devido às preocupações com a segurança e com o desempenho e pela falta de uma metodologia para o desenvolvimento das aplicações. Esta diferença, porém, tende a diminuir devido ao grande interesse na mobilidade de código motivado não somente pelos benefícios que esta pode oferecer às aplicações normalmente contidas entre os modelos tradicionais, mas também por possibilitar o desenvolvimento de aplicações inusitadas.

Através dos domínios de aplicação que tendem a explorar os benefícios desta nova área, pode-se ter uma idéia do futuro promissor das aplicações que incorporam a noção de mo-

bilidade. Entre estes, pode-se citar os domínios já introduzidos no Capítulo 1: consulta de informação distribuída, serviços de telecomunicações, gerenciamento e cooperação de atividades do tipo *workflow* e comércio eletrônico.

3.3 Agentes Móveis

O paradigma de agentes móveis vem se apresentando como uma tecnologia emergente para o desenvolvimento de aplicações em ambientes abertos, distribuídos e heterogêneos, como a Internet. O termo **agente** tem sido empregado em diversas disciplinas da ciência da computação, incluindo inteligência artificial, arquitetura de objetos distribuídos, sistema de aprendizado adaptativo, sistema especialista, algoritmos genéticos e em estudos sobre ambientes sociais *on-line* e colaborativos. Sundsted (1998) afirma ser difícil encontrar uma definição sucinta para agentes que inclua todas as características que muitos pesquisadores e desenvolvedores consideram que os agentes apresentam, e excluir ao mesmo tempo tudo o que eles não são. Stan Franklin, em *Is it an Agent, or just a Program ? A taxonomy for Autonomous Agentes* (Franklin e Graesser, 1996), aborda essa diversidade e formaliza uma definição para agentes, que os diferencia de simples programas. Segundo o autor, agentes são, por definição, programas que estão situados dentro de um ambiente de execução que necessitam ter algumas propriedades para serem agentes. A tabela 3.2 apresenta estas propriedades.

Propriedades	**Outros Nomes**	**Significado**
1. Reativos	-	Percebe as mudanças no ambiente e atua de acordo com essas mudanças
2. Autônomos	-	Tem o controle sobre suas próprias ações
3. Orientado a objetivos	Pró-ativo	Não responde simplesmente em resposta ao ambiente
4. Temporalmente contínuo	-	É um processo continuamente em execução
5. Comunicativo	Sociável	É capaz de se comunicar com outros agentes
6. Adaptativo	Capaz de aprender	Muda o seu comportamento baseado em suas experiências anteriores
7. Móvel	-	Possui habilidade para se transportar de uma máquina para outra
8. Caráter	-	Possui personalidade e estado emocional

Tabela 3.2: Propriedades de um Agente

Por definição, todo agente deve satisfazer as quatro primeiras propriedades da tabela 3.2 (em negrito). As outras quatro são consideradas características ortogonais. Como mobilidade é uma característica ortogonal de agentes, nem todos os agentes necessitam ser móveis. Um agente pode apenas se comunicar através de meios convencionais que incluem várias formas de chamadas de procedimento remoto (RPC) e trocas de mensagens. Esses agentes são

conhecidos como **estacionários**. Segundo a OMG (OMG, 2000), um agente estacionário se executa apenas sobre o sistema onde este iniciou a execução. Se o agente necessita de informações que não estão no seu sistema, ou necessita interagir com um agente sobre um sistema diferente, o agente usa tipicamente mecanismos de comunicação, como RPC. Em contrapartida, um agente móvel não está limitado ao sistema em que este inicia sua execução. O agente móvel é livre para viajar entre vários sítios em uma rede de computadores (OMG, 2000).

Conforme definido no Capítulo 1, um agente móvel é considerado neste trabalho um agente de *software* que migra de um sítio para outro em uma rede heterogênea em nome de um individuo ou organização, que se executa de maneira autônoma no seu destino.

Geralmente, os agentes são programados em uma linguagem interpretada ou baseada em *scripts* (por exemplo, Tcl, Java e Python) para garantias de portabilidade. Cada agente tem a sua própria *thread* de execução para que tarefas possam ser executadas segundo sua própria iniciativa.

3.3.1 Conceitos Fundamentais

Quando um agente viaja, seu estado e código são também transportados. Neste contexto, o termo **estado do agente** pode ser seu estado de execução e valores de atributos do agente que determinam o que deve ser feito quando a execução for retomada no ambiente computacional destino. O estado de execução do agente é o seu estado em tempo de execução, incluindo o contador do programa e *frame stacks*. Os valores de atributos do agente incluem a informação de controle do sistema de agente associado (por exemplo, o tempo de vida do agente). O termo **código** pode ser entendido, no contexto da orientação a objetos, como as classes necessárias para o agente se executar.

A OMG (Object Management Group) publicou em 1998 um documento que trata da interoperabilidade entre sistemas de agentes móveis chamado MASIF- *Mobile Agent Specification Interoperability Facility*. A versão atualizada deste documento passou a se chamar MAF- *Mobile Agent Facility* e foi publicado em 2000 (OMG, 2000). Neste documento, a OMG apresenta um modelo conceitual comum que será adotado para definir conceitos básicos relativos ao paradigma de agentes móveis. São eles:

Autoridade do Agente. Identifica a pessoa ou organização para quem o agente atua (proprietário do agente). Vale ressaltar que uma autoridade deve ser autenticada.

Nomes de Agentes. Agentes requerem nomes que permitem que estes sejam identificados nas operações de gerenciamento, e podem ser localizados através de um serviço de nomes. Uma identidade de agente é um valor único dentro do escopo da autoridade que identifica uma instância do agente particular.

Sistema de Agentes ou Plataformas de Agentes. É uma plataforma que pode ser usada para criar, interpretar, executar, transferir e retomar agentes [1]. Como um agente, um sistema de agentes está associado a uma autoridade que identifica a pessoa ou organização para quem o sistema de agentes atua. Um sistema de agentes é identificado pelo seu nome e endereço. Um sítio pode conter um ou mais plataformas de agentes. Um tipo de sistema de agentes descreve o perfil (profile) de um agente. Por exemplo, se o tipo de sistema é o *Aglet*, o sistema de agentes, que é implementado pela IBM e suporta Java como linguagem de agentes, tem um itinerário[2] para viajar e usa Serialização de Objetos Java.

Infra-estrutura de Comunicação. Toda a comunicação entre sistemas de agentes é feita através da Infra-estrutura de comunicação (CI), conforme apresentado na Figura 3.4. Uma infra-estrutura de comunicação fornece serviços de comunicação, bem como serviço de nomes e de segurança para um sistema de agentes.

Lugar (*Place*) ou Contexto. Quando um agente se transfere, este agente viaja entre ambientes de execução chamados de lugares. Um lugar é um contexto dentro de um sistema de agentes no qual um agente pode ser executado. Este contexto pode fornecer funções, como por exemplo, controle de acesso. O lugar fonte e o lugar destino podem residir em um mesmo sistema de agentes ou em sistemas de agentes diferentes que suportem o mesmo perfil de agente. Um lugar está associado a uma localização, a qual consiste do nome do lugar e do endereço do sistema de agentes onde este lugar se localiza. Um sistema de agentes pode conter um ou mais lugares e um lugar pode conter um ou mais agentes (ver Figura 3.4).

Serialização. É o processo de conversão de um objeto dentro de um fluxo de *bytes* que pode depois ser desserializado. A desserialização é a restauração do agente a partir da forma serializada (p.ex.: um fluxo de *bytes* armazenados em um arquivo) para a forma original. A chave para o armazenamento ou para a transferência é a representação do estado de um agente em uma forma serializada que seja suficiente para reconstruir o agente [3].

***Codebase*.** Especifica a localização das classes usadas por um agente. As classes podem estar disponíveis através de um servidor Web. Se um sistema de agentes é responsável por fornecer as classes necessárias, o *codebase* deve ter as informações para localizar as classes. Neste caso um sistema de agentes é chamado de um fornecedor de classes (*class provider*).

[1] Quando relacionados à terminologia definida por Picco (1998), um agente equivale a uma unidade de execução e um sistema de agentes equivale a um ambiente computacional.

[2] Um itinerário é uma coleção de tarefas a serem executadas em uma seqüencia de localizações.

[3] A linguagem Java fornece um mecanismo de serialização que pode representar o estado de um objeto em uma forma serializada suficientemente detalhada para que depois o objeto possa ser reconstruído. A forma serializada do objeto deve ser capaz de identificar a classe Java na qual o estado do objeto foi salvo para restaurar o estado em uma nova instância.

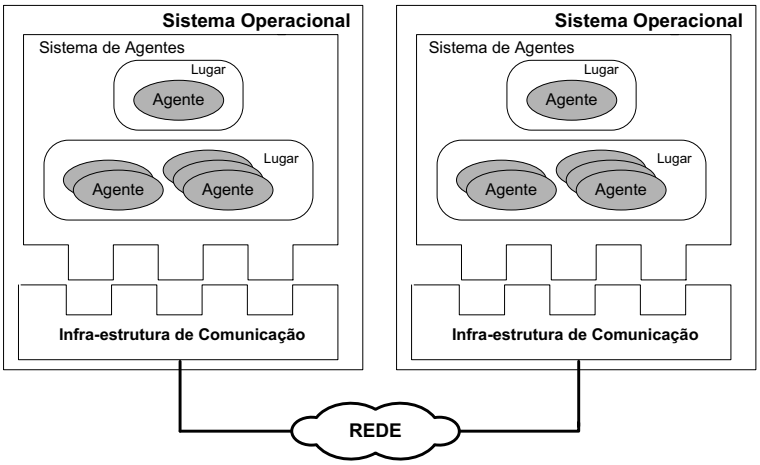


Figura 3.4: Sistema de Agentes

3.3.2 Vantagens dos Agentes Móveis

Apesar do paradigma de agentes móveis se apresentar como uma tecnologia emergente, o interesse nos agentes móveis não está motivado na tecnologia por si só, mas pelos benefícios que estes fornecem para criação de sistemas distribuídos. Lange (1998a) apresenta sete boas razões para se usar agentes móveis, a saber:

Redução da Carga da Rede. Sistemas distribuídos geralmente dependem dos protocolos de comunicação que envolvem múltiplas interações para executar uma dada tarefa, especialmente quando medidas de segurança estão embutidas. O resultado é uma grande quantidade de tráfego na rede. Os agentes móveis permitem empacotar uma negociação e transferi-la para o sítio destino, para que as interações sejam realizadas localmente (ver Figura 3.5). Quando grandes volumes de dados são armazenados em sítios remotos, os dados devem ser processados onde estes estão localizados, ao invés de movê-los pela rede.

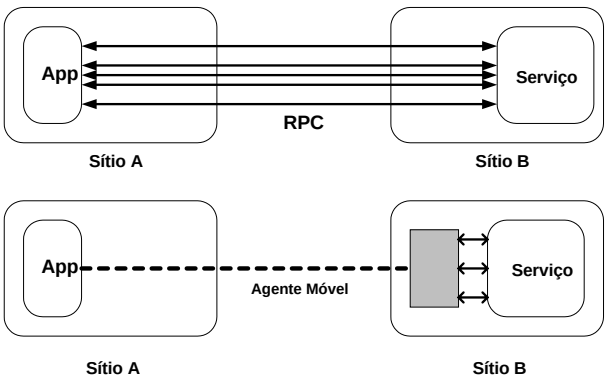


Figura 3.5: Redução da Carga da Rede

Diminuição da Latência da Rede. Sistemas de tempo real, como robôs em processos de

manufatura, necessitam responder a mudanças em seus ambientes em tempo real. Controlar esses sistemas pela rede da fábrica, que muitas vezes é de tamanho considerável, envolve uma latência significativa. Para muitos sistemas de tempo real, essa latência não é aceitável. A tecnologia de agentes móveis oferece uma solução para o problema da latência uma vez que estes podem ser disparados por um computador central para atuar localmente e executar diretamente as instruções do controlador.

Encapsulamento de Protocolos. Quando dados são trocados em um sistema distribuído, cada sítio possui o código que implementa os protocolos necessários para que esta troca de dados aconteça. Como os protocolos estão sempre evoluindo para suportar novas funcionalidades e melhorar a eficiência, a atualização desses protocolos se torna uma tarefa complicada e pode transformar os protocolos em um problema legado. Os agentes móveis são capazes de mover-se para sítios remotos e estabelecer canais baseados em protocolos proprietários.

Execução Assíncrona e Autônoma. Geralmente, dispositivos móveis (por exemplo, PDA e celular) dependem de conexões de rede frágeis e caras. Ou seja, tarefas que requerem uma conexão aberta e contínua entre dispositivos móveis e uma rede fixa é quase econômica e tecnicamente não factível. Essas tarefas podem ser embutidas dentro de um agente móvel que pode ser transferido pela rede. Depois de despachado, o agente se torna independente do processo que o criou e pode operar de forma assíncrona e autônoma (ver Figura 3.6). O dispositivo móvel pode ser reconectado a qualquer momento para coletar o agente. A habilidade de um agente móvel em operar desconectado de uma máquina cliente é uma característica essencial para aplicações baseadas em computação móvel e permite a independência das características dos canais de comunicação, tais como atrasos e falhas, garantindo assim a realização de tarefas sem necessitar de garantias de qualidade do canal de comunicação (Gray *et al.*, 2000; Schmidt, 2003).

Adaptação Dinâmica. Os agentes móveis têm a habilidade para sentir o seu ambiente de execução e reagir autonomamente às mudanças. Agentes móveis múltiplos possuem a habilidade de se distribuirem entre os sítios da rede de forma a manter uma configuração ótima para resolver um problema particular, podendo auxiliar o balanceamento de carga quando percebem que o ambiente de execução torna-se sobrecarregado.

Naturalmente Heterogêneos. A computação na rede deve ser fundamentalmente heterogênea, tanto de hardware quanto de software. Como os agentes móveis são geralmente independentes da camada de transporte e do computador, e dependem apenas de seu ambiente de execução, estes fornecem condições ótimas para a integração de sistemas.

Robustos e Tolerantes a Falhas. A habilidade dos agentes móveis em reagir dinamicamente em situações e eventos desfavoráveis torna fácil construir sistemas distribuídos robustos e tolerantes a falhas. Se um sítio estiver saindo do ar, todos os agentes em execução na máquina podem ser alertados para serem despachados para outro sítio da rede para continuar a sua operação.

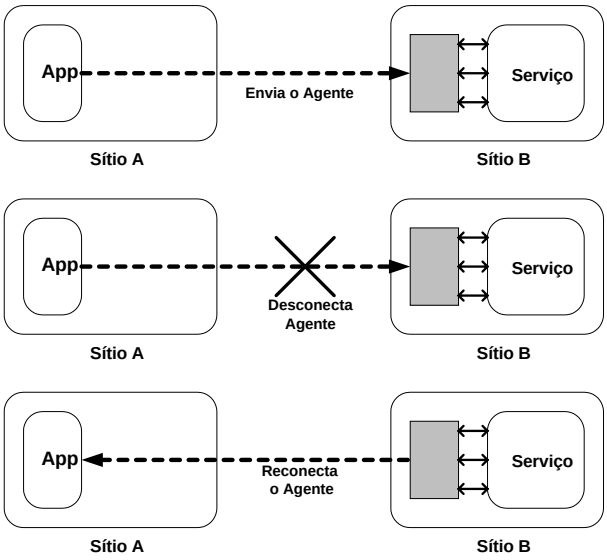


Figura 3.6: Execução Assíncrona e Autônoma

Gray *et al.* (2000) apresentam ainda outra vantagem atribuída ao uso de agentes móveis:

Implantação Dinâmica. Os agentes móveis possibilitam ainda a implantação dinâmica de novas funcionalidades ao sistema ou à atualização destas. Na implantação dinâmica, uma aplicação (agente móvel) instala dinamicamente um componente de software em uma máquina remota, para então invocar este componente como se este fosse um serviço pré-instalado na máquina remota. Por exemplo, quando um cliente deseja invocar uma operação que não é disponibilizada por um servidor, este cliente pode implementar esta funcionalidade em um agente móvel e enviá-lo ao servidor para que a operação seja realizada com os recursos do próprio servidor.

3.3.3 Desvantagens dos Agentes Móveis

Conforme apresentado nas seções anteriores, os agentes móveis podem facilitar a programação de diferentes tipos de aplicações distribuídas. Entretanto, este paradigma também adiciona problemas e limitações que precisam ser consideradas quando aplicações baseadas em agentes móveis pretendem ser desenvolvidas. Algumas desvantagens da utilização de agentes móveis incluem a necessidade de serviços extras nas máquinas onde esses códigos irão se executar, assim como as preocupações com a segurança, com a robustez e com a interoperabilidade. O uso de agentes móveis requer que cada sítio cooperante forneça facilidades para executá-los. Além disso, os agentes móveis devem ser flexíveis o bastante para executarem suas tarefas sobre diferentes tipos de ambientes de execução, como por exemplo, PDAs, *Handhelds* e telefones celulares, de forma que a execução desses agentes não deve ser afetada por características particulares a estes ambientes.

A **segurança** é considerada a maior preocupação no projeto de sistemas baseados em agentes móveis. As plataformas de agentes são expostas a riscos de penetração do sistema

causados por agentes maliciosos (similar a vírus e cavalos de tróia). Agentes podem roubar ou destruir dados sensíveis e interromper o funcionamento normal dos sítios que hospedam as plataformas de agentes. Similarmente, os agentes móveis também precisam ser protegidos de plataformas maliciosas. Um agente pode carregar em seu estado informações sensíveis sobre o usuário que este representa, por exemplo, número do cartão de crédito, preferências pessoais para uma busca inteligente, dinheiro eletrônico, etc. Estes dados não podem ser revelados a plataformas visitadas e a outros agentes, além disso não deve ser permitido modificá-los arbitrariamente. Diversas questões de segurança têm sido estudadas pela comunidade de sistemas distribuídos já há algum tempo, entretanto, os mecanismos e as tecnologias desenvolvidas devem ser adaptados levando em conta a mobilidade de código (Vigna, 1998b). Há algumas preocupações de segurança que são específicas de sistemas baseados em agentes móveis, são elas: a proteção das plataformas contra os agentes maliciosos, a proteção dos agentes contra outros agentes e a proteção dos agentes contra as plataformas maliciosos. A problemática da segurança é tratada em detalhes nos próximos capítulos.

Quando os sistemas baseados em agentes móveis são maiores e mais complexos, o problema do gerenciamento dos agentes se torna mais pronunciado. Nesses sistemas são necessários mecanismos de nomeação e localização em larga escala. Os programadores de aplicações necessitam de primitivas confiáveis de controle de agentes para iniciar, parar e emitir comandos específicos para esses agentes. A própria plataforma de agentes deve incorporar mecanismos robustos e tolerantes a falhas para permitir que as aplicações operem sobre redes não-confiáveis (Karnik, 1998).

Um grande número de plataformas de agentes móveis tem sido recentemente implementadas comprovando o grande interesse pelo paradigma de agentes móveis. Porém, esta variedade põe em risco a interoperabilidade entre os sistemas de agentes móveis e o crescimento de aplicações de larga escala baseadas em agentes móveis. Interoperabilidade entre sistemas de diferentes fabricantes é um requisito fundamental para atender totalmente as necessidades do mercado aberto de serviços que a Internet está se constituindo. A única forma de promover a interoperabilidade e a diversidade de sistemas é padronizando alguns aspectos da tecnologia de agentes móveis (OMG, 2000; Bellavista *et al.*, 1999).

Atualmente, a entidade de padronização mais importante na área de agentes móveis é a OMG (*Object Management Group*) com a sua especificação MAF (*Mobile Agent Facility*) (OMG, 2000). A especificação MAF não trata da interoperabilidade de linguagens, mas sim da interoperabilidade entre sistemas de agentes escritos na mesma linguagem, porém de fabricantes diferentes. Este padrão abrange importantes aspectos da tecnologia de agentes móveis, tais como: gerenciamento de agentes, mobilidade, nomeação, localização, bem como abrange também os requisitos necessários para a segurança dos sistemas de agentes. Há algumas questões não abordadas pelo MAF, ou por estarem fora do seu escopo ou porque não estão amadurecidas o suficiente, como por exemplo, a comunicação entre os agentes e a autenticação de agentes móveis com múltiplos-saltos (*multi-hop*).

Outra organização, que busca a padronização de agentes para encontrar a interoperabilidade em sistemas distribuídos, é a FIPA (*Foundation for Intelligent Physical Agents*). O trabalho da FIPA está focado principalmente na interoperablidade de agentes inteligentes através da padronização da linguagem de comunicação usada entre agentes cooperantes. Além do *framework* genérico de comunicação, a FIPA também especifica protocolos de negociação e de ontologia para suportar interoperabilidade em áreas de aplicação específica, tais como: assistentes de viagens, entretenimento multimídia, manufatura e fornecedores de serviços de rede (Magedanz, 1999).

A semelhança dos esforços de padronização da OMG e da FIPA está na busca pela interoperabilidade dinâmica entre sistemas de software estáticos, como os sistemas de agentes móveis e os agentes inteligentes. Já a maior diferença entre agentes móveis, agentes inteligentes e as especificações da OMG e da FIPA é que um agente móvel geralmente usa uma linguagem de programação de baixo nível, enquanto um agente inteligente tem, tipicamente, uma linguagem de comunicação que incorpora um ato de fala e uma linguagem de conteúdo, baseada em lógica de predicados.

O valor da mobilidade precisa ser cuidadosamente analisado no desenvolvimento de aplicações, pois esta característica é útil em algumas aplicações, mas não em todas. Muitas vezes, um sistema híbrido baseado em agentes móveis e estacionários provê um melhor desempenho e é mais seguro e robusto do que um sistema baseado totalmente em agentes móveis (Kotz *et al.*, 2002).

3.4 Conclusões do Capítulo

Os paradigmas de sistemas distribuídos tradicionais são estáticos em relação à estrutura do código de seus componentes e do local onde estes são executados. O paradigma abordado neste capítulo tem por característica a mobilidade dos componentes do sistema. Um agente móvel é livre para viajar entre vários sítios em uma rede de computadores.

Este capítulo teve por objetivo esclarecer os conceitos que envolvem o paradigma de agentes móveis, tendo como base a terminologia de códigos móveis apresentada por Picco (1998) e o modelo conceitual comum de agentes móveis definido pela OMG na especificação MAF (OMG, 2000). As vantagens e desvantagens deste paradigma para o desenvolvimento de aplicações para ambientes abertos, distribuídos e heterogêneos, como a Internet, também foram apresentadas.

A mobilidade dos agentes em busca dos recursos necessários pode reduzir a comunicação na rede e assim reduzir os requisitos de largura de banda e latência (Tripathi *et al.*, 2001). Agentes podem ser usados em sistemas distribuídos, por exemplo, para: busca, filtragem e coleta de informações; para tarefas de administração dos sistemas; e para comércio eletrônico.

Dentre as desvantagens apresentadas, a que mais tem prejudicado o desenvolvimento de aplicações baseadas em agentes móveis é a preocupação com a segurança. É esta desvantagem o objeto de estudo desta tese.

Capítulo 4

Segurança de Sistemas Baseados em Agentes Móveis

4.1 Introdução

Inúmeros modelos existem para descrever sistemas de agentes (Fuggetta *et al.*, 1998; FIPA, 2000; OMG, 2000), entretanto, para discutir questões de segurança é suficiente o uso de uma estrutura simplificada, consistindo de apenas dois componentes principais (Jansen e Karygiannis, 1999): o **agente** e a **plataforma de agentes** (ou sistema de agentes[1]).

Sistemas baseados em agentes móveis proporcionam um ambiente distribuído em que aplicações pertencentes a usuários distintos, e portanto, com níveis de confiabilidade distintos, podem ser executados concorrentemente. Além disso, as plataformas que abrigam os agentes podem estar localizadas em diferentes redes com políticas de segurança, vulnerabilidades e outras características diferentes (Picco, 1998).

Farmer *et al.* (1996a) utilizam o **exemplo de agentes de viagem** para extrair alguns princípios importantes que uma arquitetura de segurança para agentes móveis deve procurar cumprir. Neste exemplo, um agente móvel é programado por uma agência de viagens para visitar sítios que executam aplicações de companhias aéreas, de locadoras de automóveis ou de cadeias de hotéis, procurando por um plano de viagem que atenda os requisitos solicitados por um cliente.

O cliente dispara um agente para o servidor da Companhia Aérea 1 para que este pesquise na base de dados de vôos da companhia. Com os resultados armazenados em seu ambiente, o agente então migra para o servidor da Companhia Aérea 2 para também pesquisar em sua base de dados de vôos. A partir dos sítios das companhias, o agente pode visitar sítios de hotéis onde este poderá ter direito a um desconto especial, por viajar na companhia aérea

[1] Alguns autores usam ainda o termo servidor de agentes.

conveniada com o correspondente hotel. O agente compara os vôos e as informações de tarifa, decide sobre o plano de viagem (com base nos requisitos definidos pelo cliente), migra para os sítios da companhia aérea e do hotel escolhidos e faz suas reservas.

O cliente acredita que as informações de tarifas concedidas pelas companhias aéreas e hotéis sejam verdadeiras, já que estas desejam conquistar o cliente e fechar negócios. Entretanto, a relação entre as companhias aéreas é competitiva. Essa relação ilustra um princípio crucial: **"em muitas aplicações de agentes móveis não se deve esperar que todos os participantes confiem uns nos outros"**(Farmer *et al.*, 1996a). Há inúmeros ataques que estes podem praticar. Por exemplo, o servidor da segunda companhia aérea visitada pode corromper a informação de vôo da primeira companhia (aumentando a tarifa), já que esta encontra-se armazenada no ambiente do agente. Ou este pode analisar e manipular as instruções do processo de decisão do plano de viagem em seu benefício. Diante desta possibilidade tem-se que: **"decisões cruciais de um agente devem ser tomadas em plataformas neutras, considerados confiáveis pelo emissor"** (Farmer *et al.*, 1996a).

Ainda neste cenário, outro ataque possível seria a primeira companhia aérea aumentar o número de reservas (de 2 para 100) a serem requeridas, visando enganar a segunda plataforma quando esta oferece o melhor preço. O agente irá então reservar 100 lugares da tarifa barata da segunda empresa. Com isso, usuários legítimos terão que comprar seus bilhetes na primeira companhia, já que a segunda companhia acredita que o seu vôo está lotado. Analisando este ataque, observa-se que: "**um agente móvel pode se tornar malicioso em virtude do seu estado estar corrompido"** (Farmer *et al.*, 1996a)

Analisando o exemplo dos Agentes de Viagem, pode-se concluir que a preocupação central dos sistemas que suportam agentes móveis é como estabelecer a confiança e como limitar os riscos para os sítios que hospedam as plataformas de agentes e para os agentes móveis.

O objetivo deste capítulo é analisar a problemática da segurança, considerando as ameaças de segurança que este paradigma introduz ao sistema, e apresentar uma visão geral do esquema de segurança proposto nesta tese que visa contornar a maioria dessas ameaças. Nos próximos dois capítulos, o esquema de segurança proposto será detalhado e comparado com os trabalhos relacionados apresentados na literatura.

4.1.1 Saltos de um Agente

Os saltos (*hops*) de um agente são importantes para estabelecer a confiança da plataforma em um agente e do agente em uma plataforma. Os várias saltos que um agente pode realizar dificultam o estabelecimento da confiança na plataforma e no agente. A confiança da plataforma em um agente depende da sua identidade e por onde este agente tem passado (plataformas visitadas pelo agente). A confiança do agente em uma plataforma-destino depende da identidade da plataforma e como o agente obtém a permissão da plataforma corrente para migrar para a plataforma-destino (Ordille, 1996). Por exemplo, em situações em

que ocorre autenticação mútua entre as plataformas envolvidas, a confiança do agente na plataforma-destino é maior.

Uma vez estabelecido o nível de confiança, o nível do risco para a plataforma e para o agente também pode ser estabelecido. A importância da confiança aumenta quando o controle de riscos diminui. É comum para as linguagens que produzem códigos corretos (*safety*), como Java, Safe-Tcl e Telescript, limitar os riscos de uma plataforma dependendo do agente que esta hospeda. Vale ressaltar que o nível de confiança, e também de risco, depende não apenas do dono do agente e da plataforma-destino, mas de todas as plataformas visitadas pelo agente.

Em geral, os agentes podem viajar por várias plataformas e podem ou não retornar para a sua plataforma de origem (*home*). Agentes que dão vários saltos são chamados de agentes *multi-hop*. Estabelecer a confiança para o caso de agentes *multi-hop* é difícil. Há dois casos especiais em que estabelecer a confiança é mais simples: agentes *one-hop* e agentes *two-hop* bumerangues. O agente *one-hop* salta somente do seu sítio para outra plataforma. Um agente *two-hop* bumerangue salta de sua origem para outra plataforma e depois volta. Um exemplo deste agente é o agente que visita uma plataforma, pesquisa informações e retorna à plataforma de origem com os resultados da pesquisa.

Estabelecer a confiança em agentes *one-hop* é mais simples porque a definição das responsabilidades do agente é também mais simples. O dono do agente é responsável por eventuais problemas com o agente. A plataforma que receberá o agente, por outro lado, é responsável por executar o agente com informações confiáveis e sem modificá-lo. Há diversas formas de uma plataforma receptora poder estabelecer um nível de confiança com o dono do agente. De uma forma mais geral, o código e o dado do agente podem ser digitalmente assinados pelo dono do agente e autenticados pela plataforma receptora. O dono de um agente deve especificar uma política de segurança que pode ser aplicada diretamente pela plataforma de origem. A política pode associar um nível de confiança com plataformas particulares ou com grupos de plataformas administradas por organizações particulares. Alternativamente, a política pode delegar a determinação da confiança para uma autoridade externa confiável. A plataforma de origem autentica a plataforma receptora para o agente e estabelece o nível apropriado de confiança para a plataforma baseado na política de segurança. Estabelecer a confiança nos agentes *two-hop* bumerangue é similar ao que ocorre com agentes *one-hop* porque os atores são os mesmos: o dono do agente e a plataforma.

Com os agentes *multi-hop*, a situação é diferente. Não se deve esperar que os agentes que realizam vários saltos conheçam todos os seus destinos previamente. Para agentes com itinerários livres, é desejável que os agentes e as plataformas de agentes possam analisar e avaliar a confiabilidade das plataformas ao longo do caminho do agente. Agentes podem fazer suas análises, procurando e avaliando novos sistemas de agentes para inclusão em seu caminho. Especificações de políticas especiais podem ajudar as plataformas a automatizar a aplicação de políticas de risco e de políticas de confiança para os agentes (Ordille, 1996).

Uma forma de qualificar o nível da confiança e de risco é avaliando a história dos saltos. Detalhes dessa problemática serão tratados nos capítulos 5 e 6.

4.2 Categorias de Ameaças de Segurança e Proteções Necessárias

Existem inúmeras ameaças de segurança em um sistema de agentes móveis. O Instituto Nacional de Padrões e Tecnologia dos EUA (NIST) identificou quatro categorias de ameaças (Jansen e Karygiannis, 1999)(ver Figura 4.1: (1) ameaças dos agentes atacando uma plataforma de agentes; (2) uma plataforma de agentes atacando um agente; (3) agentes atacando outros agentes em uma mesma plataforma de agentes e, (4) outras entidades atacando os agentes e/ou as plataformas de agentes. A última categoria inclui os casos de um agente atacando um agente de outra plataforma, e de uma plataforma de agentes atacando outras plataformas, uma vez que estes ataques estão focados sobre a capacidade de comunicação da plataforma para explorar vulnerabilidades potenciais. Esta última categoria também inclui muitos ataques convencionais contra o sistema operacional subjacente da plataforma de agentes.

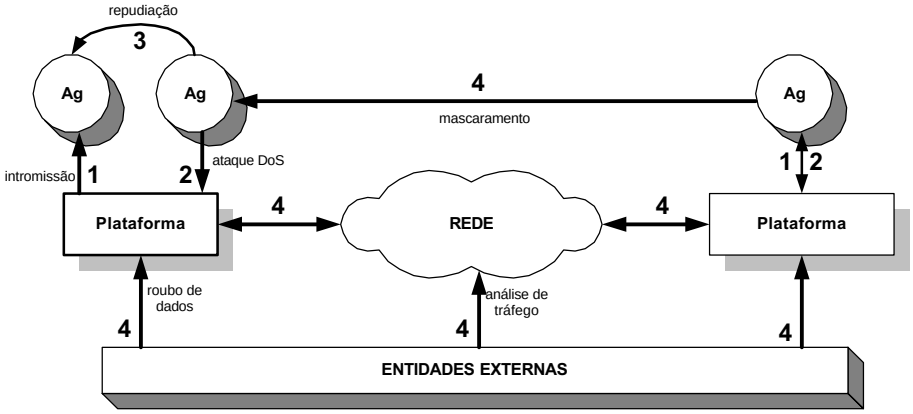


Figura 4.1: Categorias de Ameaças de Segurança

Em Vitek *et al.* (1997), os autores propõem que mecanismos de proteção sejam colocados em todas as fronteiras dos componentes de uma arquitetura de agentes móveis com o objetivo de controlar e regular as interações entre os componentes. Ou seja, visando contornar as categorias de ameaças apresentadas na Figura 4.1, os mecanismos de segurança devem ser direcionados para proteção do canal de comunicação, das plataformas de agentes e dos próprios agentes. Analisando essas categorias de ameaças, constata-se que esses mecanismos de proteção devem assegurar que:

- o acesso aos recursos locais sejam controlados para a proteção dos sítios (seta 2 e 4 da Figura 4.1);
- a plataforma de agentes seja protegida de computações maliciosas e vice-versa (setas 1 e 2 da Figura 4.1);

- os agentes sejam autenticados e possuam direitos de acesso;
- a comunicação entre as plataformas de agentes sejam protegidas, garantindo a integridade e confidencialidade do estado e do código do agente (seta 4 da Figura 4.1) ;
- os agentes se protejam mutuamente para prevenir que um agente obtenha acesso a informações privilegiadas (seta 3 da Figura 4.1).

Muitos mecanismos de segurança convencionais, usados em aplicações convencionais (p.ex.: aplicações no estilo cliente-servidor), também têm utilidade como medidas de proteção dentro do paradigma de agentes móveis. Além disso, existem hoje diversas extensões destas técnicas convencionais e técnicas projetadas especificamente para controlar códigos móveis e conteúdos executáveis (p.ex.:*applets* Java) para garantir a segurança de agentes móveis (Jansen e Karygiannis, 1999; Chess, 1998).

As ameaças pertencentes à categoria (4) da Figura 4.1 são problemas inerentes aos sistemas distribuídos. Em todo ambiente de rede há pessoas não-autorizadas que podem espionar o tráfego da rede ou se introduzirem em máquinas a fim de acessarem ou até modificarem as informações. Estas ameças e os mecanismos de segurança serão analisados na seção 4.3. As seções, a seguir, apresentarão as ameaças que agentes e plataformas de agentes estão sujeitos.

4.2.1 Ameaças dos Agentes contra as Plataformas de Agentes

Serão abordadas, nesta seção, as ameaças nas quais os agentes exploram as fraquezas de segurança de uma plataforma de agentes ou planejam ataques contra essa plataforma. Este conjunto de ameaças inclui, de acordo com Jansen e Karygiannis (1999):

- **Mascaramento**. Ocorre quando um agente se passa por um agente autorizado no esforço de ganhar acesso a serviços e recursos locais que não são atribuídos a este. Um agente mascarado pode destruir a confiança que um agente legítimo estabeleceu em uma comunidade de agentes e prejudicar sua reputação.
- **Negação de Serviço**. Surge quando os agentes lançam ataques para consumir uma quantidade excessiva de recursos computacionais de uma plataforma de agentes (p.ex. espaço em disco, CPU e portas de rede). Esses ataques de negação de serviço podem ser intencionais, explorando certas vulnerabilidades, ou não intencionais através de erros de programação.
- **Acesso Não-Autorizado**. Por exemplo, surge quando um agente tem acesso de escrita e leitura de dados nos quais este não tenha autorização, incluindo acesso a dados residuais que podem ser armazenados em um *cache* ou em outro modo de armazenamento

temporário. Todo agente que visita uma plataforma deve estar sujeito à política de segurança da plataforma.

- **Repudiação**. Ocorre quando um agente, ou seu proprietário, nega que uma transação ou comunicação com um plataforma de agentes ocorreu. Uma plataforma de agentes não pode prevenir um agente de repudiar uma transação, mas as plataformas podem garantir a disponibilidade de evidências suficientemente fortes para suportar a resolução dos desacordos.

As pesquisas em torno da **proteção de plataformas de agentes contra ataques de agentes móveis** já proporcionam alguns resultados satisfatórios tais como o uso de assinaturas digitais do código do agente e de domínios isolados de execução para agentes móveis. Alguns mecanismos já implantados nas plataformas comerciais e acadêmicas de agentes móveis apresentam uma certa eficiência, aceita, de um modo geral, na literatura. Porém, as técnicas de proteção precisam ser aperfeiçoadas, em especial, quando agentes *multi-hop* percorrem diversas plataformas em um sistema de larga escala. Para a proteção de plataformas de agentes contra agentes maliciosos, a assinatura do código do agente não é suficiente para se estabelecer a confiança neste agente. Como os agentes móveis podem se tornar maliciosos em virtude de seus estados terem sido modificados, o grau de confiança nas plataformas visitadas também é importante para a autenticação dos agentes móveis. Os mecanismos de segurança que visam proteger as plataformas de agentes encontrados na literatura, bem como o esquema de segurança proposto nesta tese, serão discutidos em detalhes no Capítulo 5.

4.2.2 Ameaças entre Agentes Móveis

A comunicação agente-para-agente pode ser diretamente entre dois agentes ou pode requerer a participação de uma plataforma subjacente. Dois ou mais agentes em um mesmo ambiente computacional devem estar protegidos um dos outros de forma a impedir a exploração de suas fraquezas de segurança. As ameaças incluem (Jansen e Karygiannis, 1999):

- **Mascaramento**. Ocorre quando um agente tenta disfarçar sua identidade no esforço de iludir o agente com quem este está se comunicando. Fazer se passar por outro agente prejudica tanto o agente que está sendo iludido quanto o agente cuja identidade está sendo assumida, especialmente em uma sociedade de agentes onde a reputação é avaliada e usada como um meio para estabelecer confiança.

- **Negação de Serviço**. Surge quando um agente lança ataques de negação de serviço sobre outros agentes. Por exemplo, enviando mensagens repetidamente para outros agentes. Linguagens de comunicação de agentes e políticas de conversação devem garantir que um agente malicioso não ocupe outro agente em um *loop* de conversação infinita com a única finalidade de deixá-lo indisponível. Agentes maliciosos podem

distribuir intencionalmente informações falsas ou inúteis para impedir outros agentes de completar corretamente suas tarefas.

- **Repudiação**. Ocorre quando um agente, participando de uma transação ou comunicação com outro agente, afirma que uma transação ou comunicação nunca ocorreu. A causa para repudiação pode ser deliberada ou acidental. Desacordos podem aumentar não apenas quando um agente repudia falsamente uma transação, mas também porque processos de negócios imperfeitos podem conduzir a diferentes visões (constatações) dos eventos. Como um agente pode repudiar uma transação como sendo o resultado de um engano, é importante que os agentes e as plataformas de agentes envolvidas na transação mantenham registros para ajudar a resolver qualquer disputa.

- **Acesso Não-Autorizado**. Se a plataforma de agentes tem mecanismos de controle fracos ou não os tem, um agente pode interferir diretamente em outro agente, invocando seus métodos públicos (tentando gerar uma sobrecarga) ou acessando e modificando os dados dos agentes ou seu código. A modificação do código do agente é uma forma insidiosa de ataque, uma vez que esta modificação pode mudar radicalmente o comportamento do agente. Um agente pode também obter informações sobre outras atividades do outro agente, usando os serviços da plataforma para espionar suas comunicações.

A **proteção entre os agentes móveis** é geralmente obtida através de mecanismos de controle de acesso que podem ser internos ou externos (Picco, 1998). Mecanismos internos permitem que um agente examine as requisições de serviços e conceda explicitamente acesso, tendo como base a identidade do agente solicitante e os parâmetros do serviço. Mecanismos de segurança externos limitam as ações de um agente com relação a outro agente, usando a política de segurança da plataforma associada com as interações dos agentes. Quando um agente móvel está se executando em uma plataforma, este pode ser visto como um recurso desta plataforma e desta forma deve ser protegido com os mesmos mecanismos adotados para proteção dos dados sensíveis de uma plataforma. Esta problemática também será analisada no Capítulo 5 e mecanismos de segurança serão propostos para proteção entre os agentes móveis.

4.2.3 Ameaças de Plataformas Maliciosas contra Agentes Móveis

Um dos problemas de segurança mais difíceis nos sistemas baseados em agentes móveis encontra-se nos perigosos ataques dos ambientes computacionais realizados contra os agentes. Para executar um agente, o ambiente computacional deve acessar o código do agente e o seu estado de execução. Por isso, é difícil proteger os agentes de sítios e/ou plataformas maliciosas. As seguintes ameaças são identificadas neste cenário (Jansen e Karygiannis, 1999):

- **Mascaramento**. Surge quando uma plataforma de agentes se passa por outra plataforma no esforço de enganar um agente móvel sobre o seu verdadeiro destino e domínio de segurança e com isso atrai os agentes para a plataforma com o intuito de extrair informações sensíveis.

- **Negação de Serviço**. Uma plataforma maliciosa pode ignorar requisições de serviço de um agente e introduzir atrasos inaceitáveis a execuções críticas; pode também simplesmente não executar o código do agente, ou encerrar o agente sem notificação. Agentes que esperam por resultados de um outro agente sobre uma plataforma maliciosa podem ser levados a estados de *deadlock*. Um agente pode entrar também em um estado de *livelocked* se uma plataforma maliciosa criar uma situação na qual algum estágio crítico de execução não consiga encontrar um objetivo e permaneça sempre realizando alguma ação secundária.

- **Intromissão ou** *Eavesdroping*. É uma ameaça clássica que envolve a interceptação e o monitoramento de uma comunicação secreta. Esta ameaça em sistemas baseados em agentes móveis é mais grave já que uma plataforma de agentes pode monitorar tanto a comunicação, como a execução de agentes. Todos os códigos e dados de origem não cifrados e os dados, subseqüentemente gerados, são apresentados à plataforma. Devido a isto, os ataques tomam a forma de violações da propriedade intelectual (revelação de algoritmos proprietários, segredos de transação, estratégias de negociação e outras informações sensíveis).

- **Modificação Não-Autorizada**. Ocorre quando um agente visita uma plataforma maliciosa que modifica seu código e/ou estado. Uma vez que um agente pode visitar diversas plataformas sobre vários domínios de segurança, mecanismos devem ser colocados para garantir a integridade do código e do estado do agente. Ou seja, é preciso proteger contra a inserção não autorizada de dados que serão encapsulados pelo agente, contra a destruição ou alteração de parte ou de todo o código e/ou estado do agente.

A **segurança dos agentes móveis contra as ameaças de plataformas maliciosas** é um problema difícil de ser contornado e ainda sem solução adequada. Isto ocorre devido ao fato de um agente ser completamente suscetível à plataforma e de ser difícil evitar a ocorrência de comportamentos maliciosos. Em Chess (1998), há afirmação de que uma plataforma que executa um dado agente tem total controle sobre as ações desse programa. Não há como prevenir a plataforma de analisar o programa, de mudar o estado do agente antes de executá-lo ou de observar seus resultados. A dificuldade deste problema pode ser medida pelo pequeno número de plataformas que implementam mecanismos relacionados a estas questões. Os mecanismos de segurança para proteger os agentes móveis devem garantir a integridade e a confidencialidade do código e do estado do agente nestas plataformas, o que é bastante complexo. Ademais, quando agentes *multi-hop* são levados em consideração, esta proteção se torna ainda mais difícil de ser alcançada. A segurança dos agentes móveis será cuidadosamente discutida no Capítulo 6.

4.3 Problemas Típicos de Agentes em Sistemas distribuídos

Os sistemas baseados em agentes móveis, assim como todos os sistemas distribuídos, devem atender a alguns requisitos de segurança, entre eles a autenticação, a responsabilização, a integridade, a confidencialidade e a disponibilidade. Estes requisitos serão brevemente analisados nesta seção.

4.3.1 Autenticidade

Ao interagirem, dois ambientes computacionais devem autenticar-se mutuamente. A autenticação de entidades permite identificar unicamente cada entidade em um sistema. Além da autenticação de entidades, tem-se ainda a autenticação da origem dos dados que permite comprovar a identidade da plataforma que originou um agente ou uma determinada mensagem ou a identidade de um agente que originou uma determinada mensagem. A literatura apresenta inúmeros protocolos criptográficos para a autenticação em sistemas distribuídos que podem ser utilizados quando o paradigma de agentes móveis é considerado (Schneier, 1996; Stallings, 1998; Bishop, 2003). Em sistemas baseados em agentes móveis, a autenticação é essencial para que se possa responsabilizar agentes e/ou plataformas pelos atos cometidos, bem como para conceder direitos de acesso e para efetuar cobrança pelo uso de recursos.

4.3.2 Responsabilização (*Accountability*)

As ações de cada processo do usuário humano ou do agente sobre uma dada plataforma devem poder ser responsabilizadas (por algo ou alguém). Para tornar isto possível, cada processo ou agente deve ser identificado de uma forma única, autenticado e auditado. A responsabilização necessita manter um registro de auditoria (*logs*) dos eventos relevantes à segurança.

Agentes móveis criam rastros de auditoria (*audit trails*) através de várias plataformas, pois cada plataforma registra eventos separados e possivelmente com diferentes pontos de vistas (políticas) para um mesmo evento. No caso em que um agente deseja acessar registros distribuídos em diferentes plataformas, se torna difícil reconstruir uma seqüencia de eventos associados a esses registros. Plataformas que guardam *logs* de auditoria distribuídos devem estar habilitadas para manter um conceito global de tempo ou ordenação de eventos, o que em sistemas distribuídos, pode ser difícil. Ações de agentes são importantes considerando as questões de responsabilidades, por exemplo, para o caso de ataques de negação de serviço (Jansen e Karygiannis, 1999) e a auditoria pode contribuir para estabelecer as responsabilidades.

4.3.3 Confidencialidade

A confidencialidade deve ser garantida a qualquer dado privado armazenado sobre uma plataforma ou transportado por um agente. Plataformas de agentes devem estar habilitadas para garantir que a comunicação entre plataformas e a comunicação em uma mesma plataforma se mantenham em sigilo. Invasores podem deduzir informações sobre as atividades dos agentes, não apenas examinando o conteúdo das mensagens trocadas, mas também analisando o tráfego entre os agentes.

Agentes móveis podem também querer manter em sigilo a sua localização. Estes podem se comunicar através de um *proxy* cuja localização é publicamente conhecida. Os agentes podem decidir se sua localização estará publicamente disponível através de serviços de nomes de agentes, e as plataformas podem aplicar diferentes políticas de segurança sobre os agentes que escolheram ser anônimos (Jansen e Karygiannis, 1999).

Como os registros de auditoria (*logs*) mantêm as gravações detalhadas das atividades de um agente sobre a plataforma, o conteúdo desses registros deve ser cuidadosamente protegido e mantido em sigilo. Registros de auditoria de agentes móveis podem ser distribuídos através dos domínios de segurança, com políticas especiais de auditoria. Por isso, alguns agentes podem querer transportar certas partes desses registros para referências futuras. Em alguns casos, o agente móvel pode solicitar à plataforma para que esta assine os registros pelos quais ela é responsável.

Para garantir a confidencialidade da comunicação, evitando que pessoas não-autorizadas compreendam o conteúdo das mensagens e dos agentes enviados, tem-se uma gama de soluções divididas entre algoritmos simétricos ou de chave secreta (e.g. DES) e assimétricos ou de chave pública (e.g. RSA).

4.3.4 Integridade

Analisando a problemática das ameaças das categorias (1) e (4) da Figura 4.1, observa-se que se faz necessário garantir a consistência dos dados armazenados em uma plataforma ou que são transportados por um agente, em particular, prevenindo a modificação não autorizada ou a simples destruição desses dados. Por exemplo, os registros de auditoria, mencionados no item 4.3.2, precisam ser protegidos de forma que permaneçam íntegros.

Mecanismos que asseguram a integridade são basicamente algoritmos de *digest* ou *checksum* (por exemplo, o MD5). Estes são fundamentais na garantia de que as informações não foram modificadas, seja por má intenção ou mesmo por um erro no processo de transmissão.

4.3.5 Disponibilidade

A plataforma de agentes deve estar habilitada para garantir a disponibilidade de dados e serviços para os agentes locais e remotos. A plataforma de agentes deve fornecer concorrência controlada, suporte a acessos simultâneos, gerenciamento de *deadlocks* e acesso exclusivo, conforme requerido. Plataformas devem estar habilitadas a conviver com falhas de *hardware* e *software* no sistema (Jansen e Karygiannis, 1999). Enquanto as plataformas podem fornecer algum nível de tolerância a faltas e recuperação de erros, agentes devem possuir mecanismos responsáveis pela própria recuperação de erros ou mesmo o tratamento de falhas.

Plataformas de agentes devem assegurar os pedidos de milhares ou milhões de visitantes e de seus agentes remotos, ou então, o risco de uma negação de serviço não-intencional é gerado. Caso uma plataforma não possa assegurar sua computação ou a carga da comunicação, esta plataforma deve fornecer uma degradação de seus serviços e notificar aos agentes a impossibilidade de fornecer o nível e a qualidade de serviços esperados pelos agentes que estão solicitando esses serviços.

A ameaça de um ataque de negação de serviço, sendo lançado por um agente malicioso, exige um controle e monitoramento dos recursos da plataforma. Um ataque de negação de serviço contra a plataforma é um ataque indevido a todos os agentes que dependem desta plataforma (Jansen e Karygiannis, 1999).

Garantir a integridade e a confidencialidade interfere na disponibilidade e também apresenta custos de desempenho. Mensagens e agentes cifrados em trânsito podem impor um atraso inaceitável no ambiente onde um suporte de tempo real é requerido. Garantir a responsabilização pode também colocar restrições sobre a disponibilidade dos serviços da plataforma.

4.4 Discussão das Questões de Segurança no Projeto de Aplicações Baseadas em Agentes Móveis

Segundo Jansen e Karygiannis (1999), são diversos os impactos provocados com o uso de mecanismos de segurança para proteger as aplicações baseadas em agentes móveis. É preciso avaliar o uso de mecanismos de segurança ainda na fase de projeto de uma aplicação já que estes podem ditar decisões de projeto e, principalmente, porque estes podem negar os benefícios do uso dos agentes móveis. Em (Jansen e Karygiannis, 1999), algumas questões de segurança são discutidas, tendo em vista os benefícios do paradigma de agentes móveis apresentados na seção 3.3.2. Esta análise do impacto da segurança é sintetizada a seguir.

Soluções de agentes móveis têm sido propostas para sistemas que necessitam responder a mudanças no ambiente em tempo real. Um exemplo é o uso de agentes móveis para con-

trolar robôs responsáveis por processos de manufatura distribuídos. Embora esta abordagem possa resolver problemas de latência da rede de computadores, esta também levanta sérias questões quanto a segurança. Logo, mecanismos de segurança precisam ser considerados. Quando os mecanismos de segurança necessários para fornecer o controle dos recursos e outras proteções são introduzidos, estes podem impor um custo de desempenho inaceitável para aplicações em tempo real. Com isso, o desenvolvedor da aplicação deve estabelecer um confronto entre a segurança e o desempenho para a escolha dos mecanismos de proteção.

Agentes móveis se mostram adequados para resolver problemas de pesquisa e análise envolvendo múltiplos recursos distribuídos. Uma abordagem de busca e análise de dados, baseada em agentes móveis, pode ajudar a diminuir a carga da rede. Entretanto, esses benefícios não podem ser considerados, a menos que mecanismos de segurança sejam adicionados para garantir a proteção do agente e principalmente a proteção dos recursos da plataforma. Então, os benefícios do uso de agentes móveis para reduzir a carga da rede depende da disponibilidade de um esquema de segurança para execução e transferência de agentes móveis.

Muita atenção está sendo dada para o uso de agentes móveis com dispositivos móveis como telefones celulares, PDAs e equipamentos militares. Sua execução assíncrona e autônoma os tornam ferramentas desejáveis para aplicações que usam conexões de rede caras e frágeis. Embora um agente móvel possa ter autonomia de movimento e não precisar estar conectado à sua plataforma de origem, este, por necessidade de serviços de segurança, pode depender do sítio que o iniciou. Por exemplo, é o caso de um agente móvel que faz parte de uma aplicação de comércio eletrônico e que precisa assinar digitalmente documentos relacionados com uma transação. Como este não pode transportar consigo de forma segura a sua chave privada, o agente deve confiar na plataforma de origem ou em uma terceira parte confiável para fornecer este serviço. Neste exemplo, embora o agente tenha autonomia de movimento, a necessidade de uma plataforma confiável para serviços de segurança limita os tipos de tarefas que este pode executar autonomamente.

Agentes móveis têm a habilidade de sentir seu ambiente de execução e reagir autonomamente às mudanças. Estes podem migrar para fora do seu domínio de segurança e com isso tornam-se sujeitos a políticas de segurança locais. A forma pela qual este se adapta será então influenciada e potencialmente restringida às políticas de segurança da plataforma. Comunicação, negociação e gerenciamento de políticas de segurança de agentes móveis também introduzem novos custos de administração de segurança para a plataforma de agentes móveis.

Como os agentes móveis são dependentes apenas do ambiente de execução, estes oferecem uma abordagem atrativa para integração de sistemas heterogêneos. Os benefícios da heterogeneidade introduzem, entretanto, novas preocupações de segurança. As implementações correntes das máquinas virtuais ou dos interpretadores, que tornam possíveis ambientes computacionais heterogêneos, não fornecem suporte adequado para controle de alguns recursos. Por exemplo, o Java não fornece meios para um sítio limitar a alocação de recursos

de memória e do processador para um dado objeto ou uma *thread* e é, por conseguinte, suscetível a ataques de negação de serviço.

A habilidade dos agentes móveis em reagir dinamicamente em situações e eventos desfavoráveis torna fácil a construção de sistemas distribuídos tolerantes a falhas e ao mesmo tempo robustos. Mesmo com um arsenal de técnicas existentes para fornecer segurança e tolerância a falhas, o desenvolvedor deve tomar o cuidado na seleção dos mecanismos a serem usados e de como estes interferirão no desempenho e nas funcionalidades do sistema como um todo. Desenvolvedores de plataformas de agentes móveis também precisam confrontar a segurança e a tolerância a falhas. Por exemplo, para contornar riscos de segurança envolvidos na mobilidade de agentes *multi-hop*, algumas arquiteturas têm sido construídas tendo como base um modelo cliente-servidor centralizado, requerendo que o agente retorne ao servidor central antes de se mover para outro sítio (Ad Astra Engineering, 1998). É fácil perceber que tratando os riscos de segurança desta maneira, todos os agentes móveis se tornarão vulneráveis a falhas do servidor central e assim problemas quanto à escalabilidade surgirão.

As ameaças de segurança no contexto das aplicações (distribuídas) baseadas em agentes móveis e as limitações dos mecanismos de proteção existentes comprovam a necessidade de um esquema de segurança portável, eficiente, escalável, interoperável e de fácil configuração e uso. Além disso, o esquema de segurança deve ser flexível para que um conjunto de mecanismos de proteção possa ser selecionado de acordo com os requisitos específicos de cada aplicação. A seção, a seguir, apresenta uma visão geral do esquema de segurança para o uso de agentes móveis em sistemas abertos que visa atender a estes requisitos.

4.5 Proposta de um Esquema de Segurança para Sistemas Baseados em Agentes Móveis

Conforme analisado na seção 3.3.3, as questões de segurança ainda prejudicam a aceitação do paradigma de agentes móveis para o desenvolvimento de aplicações distribuídas. As limitações são ainda maiores quando se leva em conta aplicações em sistemas abertos que anseiam por escalabilidade, portabilidade e interoperabilidade, já que estas aplicações envolvem vários domínios administrativos e tecnologias heterogêneas. Nesse contexto, as soluções para nomes e segurança devem ser sempre flexíveis, escaláveis e abertas, permitindo que estes sistemas possam evoluir durante a sua dinâmica.

Foi a constatação dessas limitações e a preocupação com estes aspectos específicos da segurança em aplicações de sistemas abertos baseados em agentes móveis que motivaram o esquema de segurança proposto, chamado de ***MASS — Mobile Agent Security Scheme***.

O esquema proposto está baseado em um modelo de agentes que permite múltiplos-saltos

(*multi-hop*) e itinerários livres. Podem ser considerados com itinerários livres os agentes descritos em quaisquer uma das seguintes situações:

- *m* plataformas são pré-determinadas para comporem o percurso do agente móvel, mas este tem somente a autonomia para escolher a seqüência destas visitas;
- o agente conhece a priori as *m* plataformas que podem vir a compor o seu percurso, mas este é livre para decidir se irá visitar todas as *m* plataformas e em qual seqüência;
- ou as *m* plataformas a serem visitadas não são conhecidas na sua partida.

A incorporação da noção de agentes móveis em sistemas de larga escala exige que as plataformas de agentes sigam padrões que visem garantir a interoperabilidade entre plataformas de diferentes fabricantes. A garantia desta interoperabilidade se configura como um requisito fundamental para atender integralmente às necessidades do mercado aberto de serviços da Internet, por exemplo. Visando possibilitar a sua implantação nesses sistemas abertos, o esquema proposto apresenta as plataformas em conformidade com a especificação MAF (*Mobile Agent Facility*) (OMG, 2000), o que garante a interoperabilidade entre as plataformas de agentes. Além disso, o ***MASS*** faz uso das redes de confiança SPKI/SDSI (Elisson, 1999; Lampson e Rivest, 1996), da infra-estrutura de chave pública do SPKI (Elisson, 1999) e do conceito de federações SPKI (Santin *et al.*, 2003) que oferecem mecanismos para a escalabilidade e flexibilidade necessária a estes tipos de aplicações.

4.5.1 Visão Geral do Esquema Proposto: *MASS*

A Figura 4.2 apresenta uma **visão geral** dos procedimentos suportados pelo esquema *MASS*, composto de técnicas de prevenção e de detecção, que visam prover segurança para o canal de comunicação, para os agentes móveis e para as plataformas de agentes. Para atender às necessidades específicas de aplicações, este esquema é flexível de modo que o mesmo pode ser especializado através da seleção de um subconjunto de mecanismos. Diretrizes para seleção dos mecanismos de segurança mais adequados para uma dada aplicação são propostas no Capítulo 7.

Programadores de agentes que desejam proteger a integridade do agente podem utilizar de técnicas para criação de agentes móveis protegidos (**passo 1**, Figura 4.2). Antes de enviar um agente móvel, a plataforma fonte pode estabelecer um canal seguro com a plataforma-destino através do protocolo SSL (Freier *et al.*, 1996), do protocolo de autenticação mútua e do processo de verificação de reputação de plataformas de agentes (**passo 2**). Esta reputação é definida por um mecanismo de controle social baseado em reputação. Antes de instanciar uma *thread* para um agente, a plataforma-destino deve então executar o processo de autenticação do agente recebido, através do autenticador *multihop*, visando estabelecer a confiança

no agente (**passo 3**). Apesar da autenticação de agentes móveis ser uma técnica que primeiramente visa proteger as plataformas, esta também é usada para detectar modificações não autorizadas em um agente no decorrer de suas viagens e para auxiliar o processo de identificação de plataformas maliciosas. Durante o processo de autenticação do agente, quando uma violação da integridade do agente for detectada, o mecanismo que aplica o controle social das plataformas deve ser acionado para que a identificação de plataformas maliciosas seja possível. Por fim, após estabelecida a confiança no agente e com base nos certificados de autorização SPKI/SDSI do agente, são estabelecidos os domínios de proteção em que estes agentes serão executados (**passo 4**).

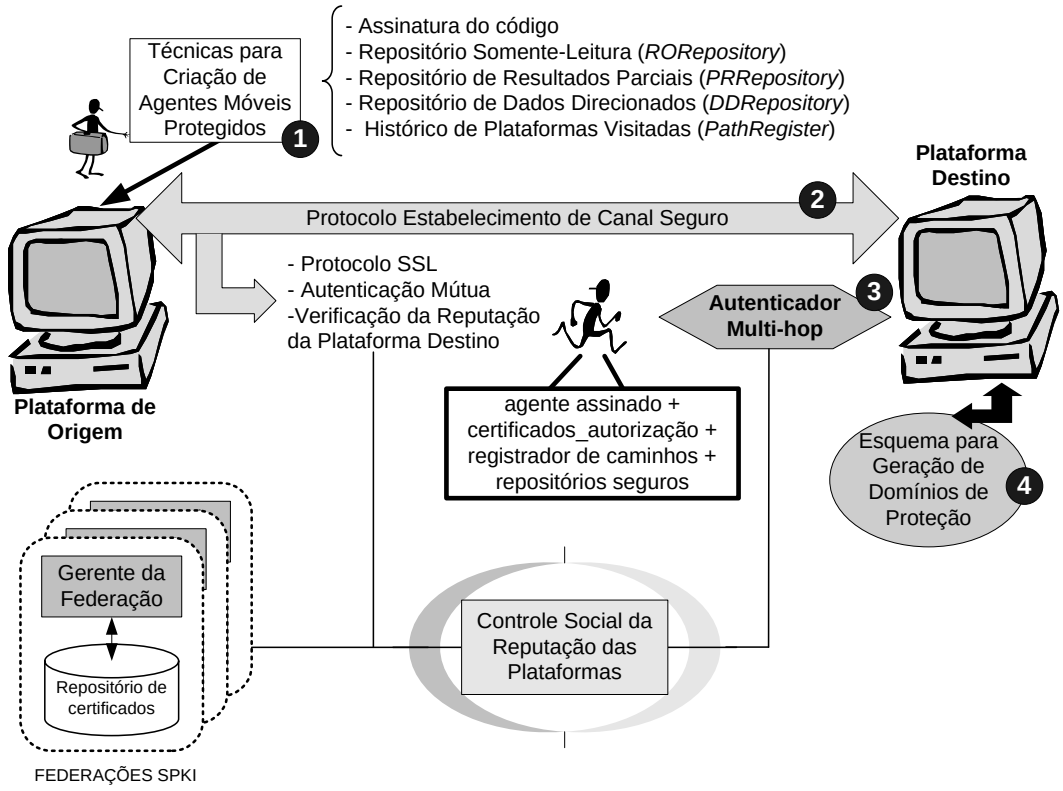


Figura 4.2: *MASS* —Esquema de Segurança para Sistemas Baseados em Agentes Móveis

Visando uma melhor comparação com os trabalhos apresentados na literatura, o esquema *MASS* foi dividido em dois sub-esquemas de acordo com o alvo de segurança: **o esquema para proteção das plataformas de agentes móveis** — $MASS_{ap}$: *Protecting **A**gents **P**latforms in MASS* — e o **esquema para proteção dos agentes móveis** — $MASS_{ma}$: *Protecting **M**obile **A**gents in MASS*. Estes sub-esquemas serão descritos em detalhes nos Capítulos 5 e 6, respectivamente.

4.5.2 Suportes Necessários, Premissas e Objetivos de Segurança do Esquema Proposto

4.5.2.1 Controle Social

Políticas de controle social, segundo Rasmusson e Jansson (1996), constituem abordagens de segurança branda (*soft security*) já que os mecanismos que as implementam não negam a existência de componentes maliciosos no sistema. Nesta abordagem, cada participante do grupo é responsável pela segurança, ou seja, não há uma entidade global ou externa ao grupo que centralize o controle da segurança. Mecanismos que implementam políticas de controle social definem o isolamento dos componentes detectados como sendo maliciosos do convívio de outros componentes corretos de um sistema. Portanto, o controle social estabelece um tipo de comportamento correto que é imposto aos membros de um grupo e cada membro fiscaliza o comportamento dos outros membros. Reputação é a imagem da organização ou entidade, a maneira como ela é vista e reconhecida por aqueles que interagem com ela, direta ou indiretamente (Rasmusson *et al.*, 1997). Um mecanismo de controle social baseado em reputação e que se utiliza da organização das federações SPKI foi definido e integrado ao esquema de segurança proposto, conforme apresentado na Figura 4.2. Detalhes sobre esse mecanismo serão apresentados no Capítulo 6.

4.5.2.2 Federações SPKI

De acordo com as suas classes de serviços, as plataformas de agentes móveis podem se agrupar constituindo **federações** de serviço e definindo relações de confiança entre si (ver exemplos na Figura 4.3). O conceito de federação, que estende o modelo de confiança do SPKI, introduzido em (Santin *et al.*, 2003), tem por objetivo localizar certificados e principais, facilitando o acesso de clientes a serviços através do compartilhamento de seus repositórios de certificados. A integração do conceito de federação ao modelo de confiança do SPKI possibilita ainda a criação dinâmica de novas cadeias de confiança, quando não há cadeias apropriadas para um acesso desejado. Estas cadeias são construídas, de forma rápida e eficiente, a partir dos certificados disponíveis nos repositórios de certificados das federações. Nesta forma de organização, um membro de uma federação pode participar de outras federações e diferentes federações podem ter vínculos entre si (relações de confiança), formando assim teias de federações de escopo global (ver o exemplo da Figura 4.3). A principal função da teia de federação é ajudar um cliente, através de seus agentes, na busca de privilégios de acesso que o liguem ao serviço desejado (outra plataforma).

As federações devem fornecer repositórios de certificados e suporte de busca de cadeias de certificados. Através da colocação dos certificados de nomes e de autorização nestes repositórios, os serviços disponíveis nas plataformas associadas podem ser divulgados. A inclusão de uma plataforma em uma destas federações deverá ser negociada com o Gerente

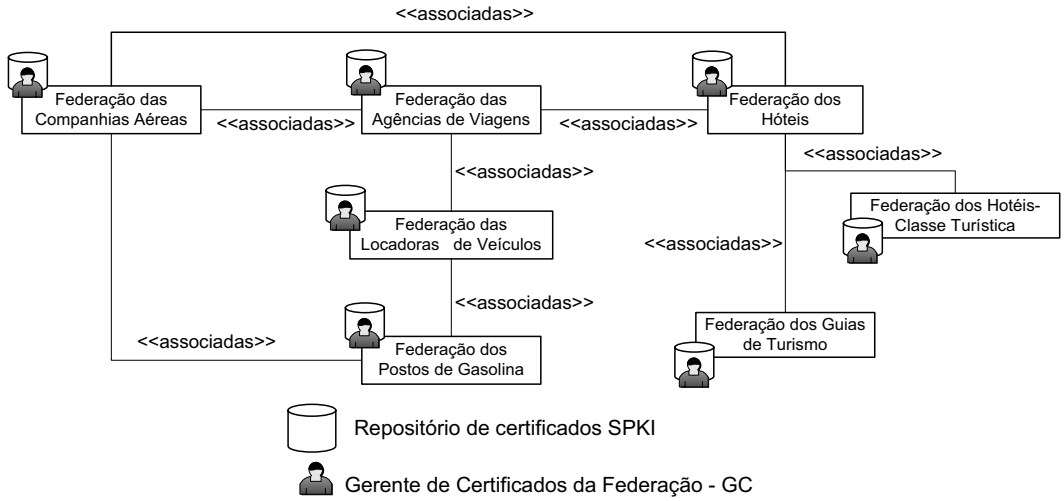


Figura 4.3: Exemplo de uma Teia de Federações SPKI

de Certificados (**GC**) que controla este serviço de armazenamento de certificados. Maiores detalhes sobre o conceito de federações podem ser obtidos em Santin *et al.* (2003).

No esquema *MASS*, as federações SPKI são usadas não só para localizar certificados e ajudar na construção de novas cadeias de certificados entre plataformas, mas também para auxiliar o estabelecimento da confiança de um agente em uma plataforma. Quando um agente é criado, uma lista indicando as federações de serviço cujas plataformas membros estão autorizadas para executar o agente, pode ser definida e anexada ao agente. Outra característica acrescida com a formação de federações de plataformas é a possibilidade de se criar e controlar dinamicamente as reputações das plataformas que integram uma determinada federação. Para isto, o **GC** de uma federação é responsável por manter uma lista com a reputação das plataformas que estão ligadas a esta federação e uma lista "*negra*" contendo informações das plataformas maliciosas que fizeram parte da federação e que foram excluídas devido a maus comportamentos. Mais detalhes sobre o uso do conceito de federações no esquema de segurança estão descritos nos Capítulos 5 e 6.

4.5.2.3 Premissas do *MASS*

O esquema proposto é construído tomando como base as seguintes premissas relativas à segurança:

- a plataforma de origem de um agente é sempre considerada confiável para este agente;
- a plataforma de origem ou outra plataforma, igualmente confiável, são implementadas de forma segura, sem falhas ou "cavalos de tróia"que possam ser explorados, e se comportam de forma não maliciosa;
- a criptografia, especialmente na forma de assinatura digital, utiliza certificados digitais através de uma infra-estrutura de chave pública ou PKI (*Public Key Infrastructure*).

- o esquema assume a não ocorrência de ataques visando a negação de serviço.

4.5.2.4 Objetivos de Segurança do *MASS*

Como uma resposta planejada às ameaças de segurança que comprometem o desenvolvimento de aplicações baseadas em agentes móveis para sistemas abertos, os seguintes objetivos de segurança foram definidos para o esquema de segurança proposto:

- prevenir a revelação não autorizada do código e do estado de um agente quando este estiver sendo enviado por uma infra-estrutura de comunicação;
- detectar a violação da integridade de um agente móvel ocorrida durante o envio de um agente por uma infra-estrutura de comunicação;
- prevenir que agentes sejam enviados e recebidos por plataformas não autenticadas;
- prevenir que agentes sejam enviados para plataformas não pertencentes às federações definidas previamente como autorizadas a receber o agente;
- registrar o caminho percorrido por um agente móvel— as plataformas visitadas por este — minimizando o problema da não repudiação;
- prevenir o acesso não autorizado de agentes móveis aos recursos locais de uma plataforma de agentes, incluindo os agentes móveis que estejam se executando nesta plataforma;
- detectar quando uma plataforma maliciosa viola a integridade do código de um agente móvel;
- detectar quando uma plataforma maliciosa viola a integridade dos dados somente-leitura carregados pelo agente;
- detectar a violação da integridade dos resultados parciais coletados e carregados por um agente;
- prevenir a revelação não autorizada dos resultados parciais coletados e carregados por um agente;
- prevenir a revelação não autorizada de dados direcionados a plataformas pré-definidas;
- aplicar um controle social baseado em reputação que visa fiscalizar e punir as plataformas de agentes móveis consideradas suspeitas de violações pertencentes a uma federação SPKI.

O esquema de segurança proposto não tem por objetivo contornar as ameaças de intromissão (*eavesdropping*) contra o código do agente, quando este visita uma plataforma maliciosa. Neste contexto, somente a proteção do estado do agente é garantida. Conforme será discutido no Capítulo 6, prevenir a revelação não autorizada do código do agente (de seus algoritmos proprietários e de suas estratégias de negociação) é um problema complexo e está fora do escopo desta tese.

4.6 Conclusões do Capítulo

A aceitação do paradigma de agentes móveis para o desenvolvimento de aplicações distribuídas ainda é prejudicada devido a questões de segurança. Devido às características do paradigma de agentes móveis e às ameaças a que este está exposto, os mecanismos de segurança devem ser projetados para proteger o suporte de comunicação, a plataforma de agentes e os próprios agentes.

Técnicas convencionais de segurança se mostram adequadas para algumas aplicações de agentes móveis. Enquanto uma ampla variedade de técnicas já existe, nem todas são compatíveis umas com as outras (não podendo ser agrupadas) devido às redundâncias no propósito, à sobreposição de funcionalidades e a não conformidade com alguns padrões. Entretanto, muitas aplicações anseiam por mecanismos ainda mais poderosos, em especial, para garantir a proteção dos agentes móveis. Os aprimoramentos necessários para incrementar as técnicas existentes incluem refinamentos que reduzem o processamento (**segurança x desempenho**) e proporcionam a combinação de mecanismos complementares para constituir esquemas de proteção mais efetivos (**segurança x compatibilidade x interoperabilidade**). Estas soluções não devem ser restritas a pequenos ambientes de aplicação (**segurança x escalabilidade**).

Uma visão geral do esquema de segurança proposto nesta tese e de seus objetivos de segurança foi apresentada neste Capítulo. A descrição detalhada dos mecanismos e procedimentos que compõem o esquema será apresentada nos próximos dois capítulos.

Capítulo 5

Proteção das Plataformas de Agentes Móveis

5.1 Introdução

Este capítulo se concentra na problemática da proteção das plataformas de agentes, considerando sistemas distribuídos de larga escala. Diante dos possíveis ataques e ações falhas de agentes móveis, os recursos e funcionalidades das plataformas de agentes devem ser protegidos. O estabelecimento de domínios isolados de execução (domínios de proteção) para cada agente visitante, combinado a um controle de acessos entre domínios, é uma abordagem que vem sendo comumente adotada com a finalidade de oferecer proteção às plataformas de agentes (Jansen e Karygiannis, 1999). Porém, a proteção contra agentes maliciosos não se restringe à limitação de seus acessos a domínios de execução nas plataformas de agentes. Outras questões precisam ser tratadas quando sistemas distribuídos de larga escala são considerados. Por exemplo, a geração destes domínios depende da autenticação e dos atributos de privilégios (credenciais) dos correspondentes agentes. Logo, quando estes sistemas de larga escala são considerados, a implantação de mecanismos de autenticação e de autorização que validam os agentes e suas credenciais por toda a dimensão do sistema se caracteriza como um grande desafio.

Neste capítulo, serão analisadas as principais técnicas encontradas na literatura para contornar essas ameaças. Após a revisão da literatura, o esquema de segurança proposto para proteção das plataformas de agentes móveis — $MASS_{ap}$ — será detalhadamente descrito e, posteriormente, comparado com os trabalhos relacionados. Desenvolvido para sistemas de larga escala, o $MASS_{ap}$ inclui um protocolo para estabelecimento de um canal seguro entre plataformas de agentes, um algoritmo para autenticação de agentes móveis visitantes e um esquema para geração de domínios de proteção isolados para execução de agentes em que este visa contornar algumas limitações do modelo de controle de acesso do Java.

5.2 Revisão da Literatura

Como já comentado, a abordagem de estabelecer domínios isolados para execução dos agentes móveis (domínios de proteção) é a técnica mais comum para proteger os recursos e funcionalidades das plataformas de agentes contra agentes maliciosos. As entradas ou mesmo trocas entre domínios são mediados por um monitor de referência. Implementações do conceito de monitor de referência têm sido realizadas desde a década de 70 e empregam diversas técnicas de segurança convencionais que são aplicáveis a ambientes de agentes móveis. Com esta abordagem, é possível prevenir que agentes interfiram na execução de outros agentes e da própria plataforma.

Com base em algumas técnicas convencionais, outras técnicas foram sendo propostas objetivando a segurança das plataformas de agentes móveis. Algumas destas técnicas são apresentadas a seguir:

- Interpretação Segura de Código;
- Código Assinado;
- Apreciação de Estado (*State Appraisal*);
- Histórico do Caminho (*Path Histories*);
- *Proof-Carrying Code*.

5.2.1 Interpretação Segura de Código

Sistemas de agentes móveis são geralmente desenvolvidos usando um *script* interpretado ou uma linguagem de programação interpretada. A principal motivação para o uso dessas técnicas de programação é permitir que plataformas de agentes forneçam toda a flexibilidade e portabilidade necessária para agentes atuarem em sistemas computacionais heterogêneos. Além disso, o alto nível de abstração conceitual, fornecido pelo ambiente interpretativo, pode facilitar o desenvolvimento dos códigos de agentes. Na abordagem de interpretação segura de código, a idéia chave é que os comandos considerados nocivos em um agente devam ser executados em um local seguro ou podem ser negados ao agente. Como exemplo, pode-se citar o Interpretador Seguro do Java (Sun, 2002) e o Safe Tcl (Levy *et al.*, 1997).

Atualmente a linguagem interpretada mais usada é a linguagem Java. A linguagem Java e o ambiente em tempo de execução possuem mecanismos que garantem um certo nível de segurança. O modelo de segurança da plataforma Java é aplicado através de vários mecanismos. Primeiramente, a linguagem foi projetada para ser segura (*safety*) em relação ao tipo de dados utilizados. Desde sua proposição inicial, o modelo é conhecido como *sandbox* (Gong, 1998). Este modelo fornece um ambiente restrito (uma "caixa de areia") para os códigos

remotos (*applets*) que são considerados não confiáveis e podem acessar somente recursos limitados. Um gerenciador de segurança (*security manager*) é responsável por determinar quais acessos aos recursos locais são permitidos (Campione *et al.*, 2000). Este modelo e a sua evolução são descritos no Apêndice A. Existem várias plataformas de agentes móveis baseadas em Java, incluindo Aglets (IBM, 1996), Ajanta (Karnik, 1998), Voyager (ObjectSpace, 1998) e Grasshopper (IKV++, 1999). Entretanto, o Java apresenta algumas limitações quanto à contabilização de recursos de memória, de CPU e de rede consumidos por *threads* individuais na execução dos agentes e ainda quanto ao seu modelo de controle de acesso estático e pouco flexível. As limitações do modelo de segurança do Java 2 também estão descritas no Apêndice A.

Provavelmente o interpretador seguro mais conhecido para linguagens baseadas em *scripts* é o Safe Tcl (Levy *et al.*, 1997), que é usado no desenvolvimento do sistema AgentTcl. O modelo Safe-Tcl oferece segurança para que programas escritos na linguagem de *script* Tcl sejam executados sem que a origem do programa seja conhecida ou sem que esta seja confiável. Este modelo está baseado em uma abordagem chamada *padded cell*. Nesta abordagem, *scripts* não confiáveis são executados em ambientes separados que estão isolados das partes confiáveis da aplicação na qual estes *scripts* se executam.

Se uma aplicação Tcl deseja executar um *script* não confiável, esta usa dois interpretadores: um interpretador mestre e um interpretador seguro ou *safe* (ver Figura 5.1). O interpretador mestre retém todas as funcionalidades (comandos) da linguagem, mas, apenas scripts confiáveis podem se executar neste. O interpretador seguro mantém apenas um conjunto de comandos que não causam qualquer dano ao sistema (base segura). Este interpretador pré-examina qualquer comando nocivo do *script* e fornece restrições nas máquinas virtuais para a execução desses *scripts* não-confiáveis. Todos os comandos considerados não seguros estão inacessíveis (escondidos) no interpretador seguro. Os comandos desabilitados incluem os comandos para acessar o sistema de arquivos, para executar sub-processos, para abrir *sockets*, entre outros. O interpretador seguro é isolado do seu interpretador mestre que não pode se comunicar diretamente com o resto da aplicação não confiável.

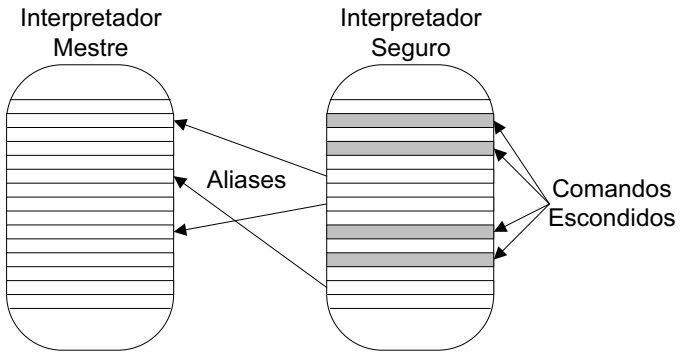


Figura 5.1: Mecanismos de Segurança do Safe-Tcl

Na prática, diversas atividades contidas nos programas são consideradas não seguras,

mas, mesmo assim, estas precisam ser realizadas para que os programas cumpram suas finalidades. Para resolver este problema, o Safe-Tcl possui um mecanismo de *alias*, análogo às chamadas em sistemas operacionais. Um *alias* é uma associação entre um comando no interpretador seguro (origem) e um comando no interpretador mestre (destino). Os comandos que estão desabilitados na base segura não foram realmente removidos do interpretador seguro, logo estes estão apenas escondidos para que não sejam invocados (Levy *et al.*, 1997). Quando um comando no interpretador seguro é chamado, o comando no interpretador mestre é executado. Diferentes políticas de segurança podem ser implementadas por diferentes conjuntos de *aliases*. Todavia, o interpretador mestre pode invocar os comandos escondidos do interpretador seguro. Isto permite ao interpretador mestre usar os comandos de forma restrita e segura a partir dos interpretadores seguros. Isto é necessário, pois muitos comandos geram resultados que só podem ser usados pelos interpretadores nos quais aqueles foram executados.

5.2.2 Código Assinado

Uma técnica fundamental para proteger plataformas de agentes é a assinatura digital do código do agente. Uma assinatura digital serve como um meio para confirmar a autenticidade de um agente (de sua origem) e a sua integridade. Tipicamente, o assinante do código é o criador do agente, o usuário do agente ou alguma entidade que tenha feito uma revisão no código do agente. Como um agente opera em favor de uma organização ou usuário final, plataformas de agentes móveis comumente usam a assinatura do código como uma indicação da autoridade na qual o agente opera (Tardo e Valente, 1996; Gray, 1996; Karjoth *et al.*, 1997; Karnik, 1998). Esta identidade autenticada é utilizada pelos mecanismos de autorização para conceder atributos de privilégios aos agentes móveis.

Uma assinatura digital é formada passando o código do agente através de uma função *hash* não-reversível, que fornece uma impressão digital ou um *digest* único do código, e cifrando o resultado com a chave secreta da autoridade do agente. Como o resumo criptográfico é único, a assinatura resultante também serve como mecanismo para verificação de integridade. Vale ressaltar que o resultado da verificação de uma assinatura pode indicar quem produziu o código e que este código não foi corrompido, mas não garante que o agente se executa corretamente (Jansen e Karygiannis, 1999).

Quando se depende unicamente da reputação de um programador do código, é prudente que se tenha uma revisão independente por uma parte confiável, por exemplo, um serviço de revisão de agentes. Em Islam *et al.* (1997), por exemplo, um serviço de avaliação de conteúdo (*content rating service*) pode ser usado para fornecer descrições alternativas e adicionais do conteúdo de um código disponível na Internet e para especificar os domínios de proteção requeridos para o conteúdo. Um domínio de proteção para um conteúdo disponibilizado na Internet pode depender de inúmeros fatores, incluindo os possíveis usuários do

conteúdo, o projetista do código, o tipo de conteúdo, a aplicação, etc. Dependendo destes fatores, este domínio poderá ser mais ou menos excludente em relação às ações de agentes.

5.2.3 Apreciação de Estado (State Appraisal)

O objetivo da técnica de apreciação de estado (Farmer *et al.*, 1996a) é garantir que um agente não tenha sido subvertido devido à alteração de suas informações de estado. A segurança do agente móvel está focada sobre as informações de estado que os agentes carregam consigo. O sucesso desta técnica depende da possibilidade de prever e contornar as alterações nocivas no estado do agente, com o uso de funções de apreciação preparadas antes do uso do agente correspondente. Funções de apreciação são usadas para determinar quais permissões são concedidas a um agente, baseadas em fatores condicionais do estado corrente e se as invariantes de estado identificadas são asseguradas (Jansen e Karygiannis, 1999). Para um agente cujo estado corrente viola uma invariante, nenhuma permissão pode ser concedida. Um agente cujo estado falha ao atender alguns fatores condicionais, segundo esta técnica, receberá um conjunto restrito de permissões.

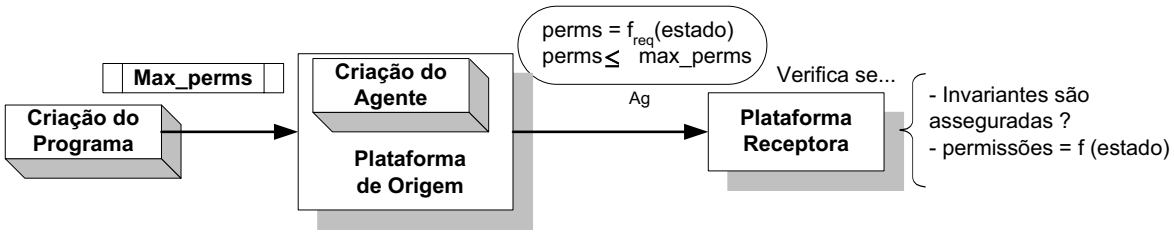


Figura 5.2: História Natural de um Agente

A Figura 5.2 apresenta a história de vida de um agente. Há três tipos de eventos na história de um agente (Farmer *et al.*, 1996a): a criação do programa, a criação do agente e a migração de um agente de um sítio de execução para outro. Na **criação do programa**, o autor irá calcular as permissões máximas (*max*) a serem atribuídas ao agente que está executando o programa em função do estado corrente do agente. Além disso, uma lista de controle de acesso pode também ser incluída para determinar quais usuários estarão autorizados a enviar o agente.

Na criação do agente que executará o programa, o proprietário do agente anexa uma segunda função de apreciação chamada função de requisição (*req*) que irá calcular as permissões que o emissor quer que um agente tenha em função do estado corrente do agente. Com isso, o proprietário do agente pode aplicar restrições de estado para reduzir as responsabilidades e ou controlar os custos do agente. O proprietário pode também incluir uma lista de controle de acesso, especificando os interpretadores que estarão autorizados a executar o agente em nome do proprietário. Quando o autor e o proprietário assinam digitalmente o agente, suas funções de apreciação são protegidas de modificações.

Uma plataforma receptora de agentes usa as funções de apreciação para analisar o estado corrente de um agente móvel visitante e para determinar quais privilégios o agente pode possuir durante a execução. Permissões são emitidas por uma plataforma, tendo como base os resultados da função de apreciação e políticas de segurança da plataforma. Nesta técnica, é difícil especificar uma regra geral para a concessão de permissões já que as funções de apreciação de estado dependem do conhecimento detalhado das aplicações. Além disso, os próprios autores afirmam ser possível que algumas modificações maliciosas não são detectadas pelas funções de apreciação (Farmer *et al.*, 1996a).

5.2.4 Histórico de Caminhos (Path Histories)

A idéia básica do Histórico do Caminho (Ordille, 1996; Chess *et al.*, 1995) é manter um registro autenticável das plataformas visitadas, a priori, por um agente, para que a próxima plataforma visitada possa determinar se processará o agente e quais as restrições de recursos serão aplicadas. A computação do histórico do caminho requer que cada plataforma visitada por um agente móvel adicione uma assinatura ao histórico, indicando sua identidade e a identidade da próxima plataforma a ser visitada para assim fornecer a história completa do caminho para a próxima plataforma (Chess *et al.*, 1995). Para prevenir alterações, a assinatura deve incluir os dados citados acima na computação do resumo criptográfico. Na recepção, a próxima plataforma pode então determinar se confia na plataforma visitada anteriormente, revendo a lista de identidades fornecidas ou a autenticidade das assinaturas de cada entrada no histórico do caminho. Esta técnica pode ser usada para proteger contra a não-repudiação, uma vez que a entrada no caminho assinado por uma plataforma é não-repudiável. Um inconveniente óbvio é que a verificação do caminho tem um custo crescente à medida que o histórico do caminho aumenta.

5.2.5 Verificação Automática de Programas

As abordagens baseadas em verificadores de código usam técnicas de provadores de teoremas para determinar se um código preserva algumas invariantes de segurança (por exemplo, o espaço de endereçamento). Dentro do contexto de agentes móveis, a abordagem **PCC** (***Proof-Carrying Code***) proposta por Necula e Lee (1998) merece destaque. O PCC é uma técnica de prevenção que associa pedaços de código do agente a provas formais que garantem a corretude do código e que são verificadas antes da execução do código. Nesta abordagem o produtor do código (p.ex.: o autor de um agente) deve provar formalmente que o programa possui propriedades de corretude (*safety*), previamente acordadas com os consumidores do código (políticas de segurança das plataformas de destino). A prova gerada deve ser codificada em uma forma que possa ser digitalmente transmitida para o consumidor do código (plataforma de destino) para que então seja validada, usando um processo confiável, automático e simples de verificação de provas.

Algumas pesquisas têm demonstrado a aplicabilidade do PCC. Por exemplo, em Colby *et al.* (2000), são descritas ferramentas para utilização do PCC com a linguagem Java e com a linguagem C que permitem provar que um dado programa não viola as características de *type-safety*, conforme definido na linguagem Java. A base da teoria do PCC está fundamentada em princípios bem estabelecidos de lógica, teoria de tipos e verificação formal. Outra importante característica desta abordagem é a independência da linguagem de programação dos agentes, podendo ser aplicada para qualquer linguagem: desde linguagens de máquina até linguagens de alto nível. Há, entretanto, alguns problemas de difícil solução, antes da abordagem ser considerada prática. Estes problemas incluem a falta de um formalismo padronizado para descrever políticas de segurança e para automatizar uma assistência para a geração de provas, e ainda, a falta de técnicas para limitar o tamanho das provas que na teoria podem aumentar de forma considerável (Necula e Lee, 1998).

5.2.6 Considerações sobre a Proteção das Plataformas de Agentes

Segundo Picco (1998), as técnicas de proteção de plataformas podem ser estáticas ou dinâmicas:

- As **estáticas** correspondem a mecanismos e regras de controle de acesso aplicados, antes da execução do agente, para determinar se o código do agente e os valores de estado em uma dada execução irão respeitar à política de controle de acesso definida no sistema. Se essa política for respeitada, então nenhuma outra verificação será feita durante a execução do agente. O PCC (Necula e Lee, 1998) é um exemplo deste esquema estático de controle de acesso, uma vez que esta técnica associa pedaços de código do agente a provas formais que garantem a corretude do código e que são verificadas antes da execução do código.

- As técnicas **dinâmicas ou em tempo de execução** atuam em ambientes computacionais implementando políticas de segurança, normalmente, na forma de um conjunto de direitos de acesso aos recursos computacionais (p.ex.: permissão de leitura e escrita de um arquivo ou do uso de uma certa quantidade de tempo da CPU). Cada tentativa de acesso aos recursos do ambiente computacional é interceptada e examinada tendo como base os direitos de acesso do agente. Desta forma, cada operação no código do agente, envolvendo acesso a recursos do ambiente, pode ser aceita ou negada. Os mecanismos clássicos de controle de acesso, citados no Capítulo 2, cumprem estes requisitos. No contexto de agentes móveis, mecanismos de controle de acesso em tempo de execução podem ser distinguidos tendo como base a informação usada para determinar o conjunto de direitos de acesso de um agente. Mecanismos baseados na autoridade determinam os direitos de acesso, tendo em vista a autoridade sobre a qual o agente está se executando. Mecanismos baseados em permissão (*permit-based*) determinam os direitos de acesso tendo em vista a credencial que está associada ao agente. Direitos

podem ser estaticamente associados ao agente para todo o seu tempo de vida ou podem ser determinados dinamicamente. Por exemplo, o mecanismo de apreciação de estado (*state appraisal*)(Farmer *et al.*, 1996a) define permissões para o agente baseadas na função associada de apreciação de estado. A função faz retornar o conjunto de direitos de acesso que o agente necessita no seu estado corrente.

Sobre a técnica de Apreciação de Estado (Farmer *et al.*, 1996a), apesar de suportar um controle de acesso em tempo de execução, essa se apresenta como uma solução limitada a alguns tipos de aplicação, já que é difícil especificar uma regra geral para concessão de privilégios devido à necessidade de um conhecimento detalhado do comportamento das aplicações. Outras abordagens que não se mostram muito eficientes e práticas de serem implantadas são as baseadas em verificação automática de programas, como, por exemplo, o PCC que implementa um controle de acesso estático fundamentado em princípios de lógica, teoria de tipos e verificação formal (Necula e Lee, 1998).

Algumas técnicas apresentadas neste capítulo já oferecem, para algumas classes de aplicação, meios que possibilitam uma segurança efetiva para as plataformas de agentes e seus recursos, em especial, quando estas técnicas podem ser combinadas. A técnica de criação de domínios isolados de execução (domínios de proteção), combinada com a técnica de assinatura de código, por exemplo, como suportada pela plataforma Java 2, possibilitam a implantação de um controle de acesso, em tempo de execução, baseado na autoridade de um agente.

Por ser um padrão *de facto* para programação de aplicações distribuídas e devido as suas boas características como linguagem de código móvel (*MCL*), a plataforma Java tem sido a *MCL* mais adotada para implementação das plataformas de agentes móveis, como por exemplo, Aglets, Concordia, Grasshopper, Voyager, Mole, SOMA, Ajanta[1]. Todas essas plataformas se utilizam do modelo de segurança do Java para implementar seus mecanismos de segurança. Entretanto, o modelo de controle de aceso do Java possui a limitação de ser estático e com características pouco flexíveis para aplicações distribuídas na definição da política de segurança. Esta limitação é discutida na seção 5.3.6. A abordagem do Histórico do Caminho (Chess *et al.*, 1995) pode auxiliar no estabelecimento da confiança em agentes *multi-hop*, já que um histórico das plataformas visitadas pode ser anexado ao agente.

Porém, estas técnicas citadas acima precisam ainda ser aprimoradas, visando garantir um controle de acesso mais adequado para aplicações em sistemas de larga escala como a Internet. Além disso, o controle de acesso baseado somente na autoridade do agente não se mostra adequado quando agentes multi-hop com itinerários livres são considerados. Uma plataforma, para se proteger de um agente, depende não só da verificação da autenticidade do proprietário do agente, mas também do grau de confiança das plataformas já visitadas pelo

[1]Estas plataformas e seus mecanismos de segurança estão descritos no Apêndice B.

agente, uma vez que um agente móvel pode-se tornar malicioso em virtude do seu estado ter sido corrompido por plataformas visitadas anteriormente (Farmer *et al.*, 1996b). Garantir a confiabilidade e a segurança dos agentes *multi-hop* com itinerários livres, já que não se conhece previamente o seu destino e as plataformas visitadas, não é tarefa fácil e ainda não possui uma solução efetiva (Ordille, 1996).

Concluindo, um esquema efetivo de autenticação e de autorização para plataformas de agentes em um sistema de larga escala deve estar baseado nos atributos de privilégios (credenciais do agente), na autenticidade do principal responsável pelo agente e na autenticidade das plataformas visitadas pelo mesmo. Mecanismos flexíveis de autenticação e de autorização que atendam aos requisitos de escalabilidade inerentes a ambientes como a Internet não são evidentes na literatura.

5.3 $MASS_{ap}$: Esquema de Segurança para Proteção das Plataformas de Agentes Móveis

O esquema de segurança proposto para proteção das plataformas de agentes móveis ($MASS_{ap}$), visando possibilitar a sua implantação em sistemas de larga escala, tem um **controle da autenticação e da autorização totalmente descentralizado**. Para isto, o esquema faz uso das cadeias de confiança SPKI/SDSI (Elisson, 1999) e do conceito de federação SPKI, definido em (Santin, 2004).

Conforme analisado no Capítulo 2, a especificação SPKI/SDSI define uma infra-estrutura de autenticação e de autorização simples e flexível baseada em certificados digitais e espaços de nomes locais. Seu modelo de confiança é igualitário, sendo que os sujeitos (ou principais) são chaves públicas e cada chave pública é uma autoridade certificadora. O mecanismo de delegação dos certificados de autorização SPKI garante a flexibilidade e escalabilidade necessária ao esquema proposto. Direitos de acesso podem ser delegados formando uma cadeia de certificados que liga um cliente (agente) a um servidor (plataforma), possibilitando uma distribuição controlada da autorização. Autorizações e permissões podem ser livremente definidas e não estão restritas a qualquer conjunto pré-definido de principais (Elisson, 1999). Devido a estas vantagens, os certificados de autorização SPKI/SDSI foram escolhidos para representar os atributos de privilégios concedidos aos agentes móveis no esquema de segurança proposto. Na estrutura proposta nesta tese, os certificados de autorização SPKI/SDSI transportados por um agente móvel definem suas credenciais.

A maior dificuldade do SPKI/SDSI está em identificar, entre os certificados de um cliente, caminhos de uma cadeia de confiança que levem a um servidor desejado. Em alguns casos, um cliente e um servidor podem nem mesmo estar conectados por uma cadeia de confiança. O conceito de federação SPKI (Santin, 2004) vem contornar esta dificuldade, auxiliando o

processo de busca e negociação de cadeias de certificados que viabilizem acessos. No esquema $MASS_{ap}$, conforme apresentado no Capítulo 3, as federações SPKI não são usadas somente para este fim, mas também para auxiliar as plataformas de agentes no estabelecimento da confiança em um dado agente e para possibilitar a implantação de um mecanismo de controle social baseado nas reputações das plataformas pertencentes a uma federação.

A Figura 5.3 apresenta os procedimentos e técnicas definidas no $MASS_{ap}$, composto de técnicas de prevenção e de detecção, que enfatizam principalmente a proteção das plataformas de agentes móveis e dos seus recursos. Neste esquema, após o processo de criação de um agente (**passo 1**), a plataforma de origem (emissora do agente) e a plataforma-destino (receptora do agente) devem passar por um protocolo de autenticação mútua para que um canal seguro seja estabelecido para o envio de agentes (**passo 2**). Em seguida, o agente deve ser enviado com as suas credenciais para ser autenticado pela plataforma-destino (**passo 3**) e então o seu domínio de execução poderá ser criado (**passo 4**). Ou seja, quando um agente chega a uma plataforma, este apresenta uma ou mais cadeias de certificados SPKI/SDSI para que o verificador da plataforma realize a checagem das cadeias de autorização. A partir destas cadeias de autorização, este verificador deverá gerar as permissões necessárias para o agente ser executado em um domínio de proteção na plataforma-destino. A geração dinâmica dos conjuntos necessários de permissões para a execução de agentes na plataforma-destino torna o esquema flexível e segue o "princípio de menos privilégio"(Saltzer e Schroeder, 1975). As Federações SPKI são necessárias para o processo de autenticação de agentes móveis (passo 3).

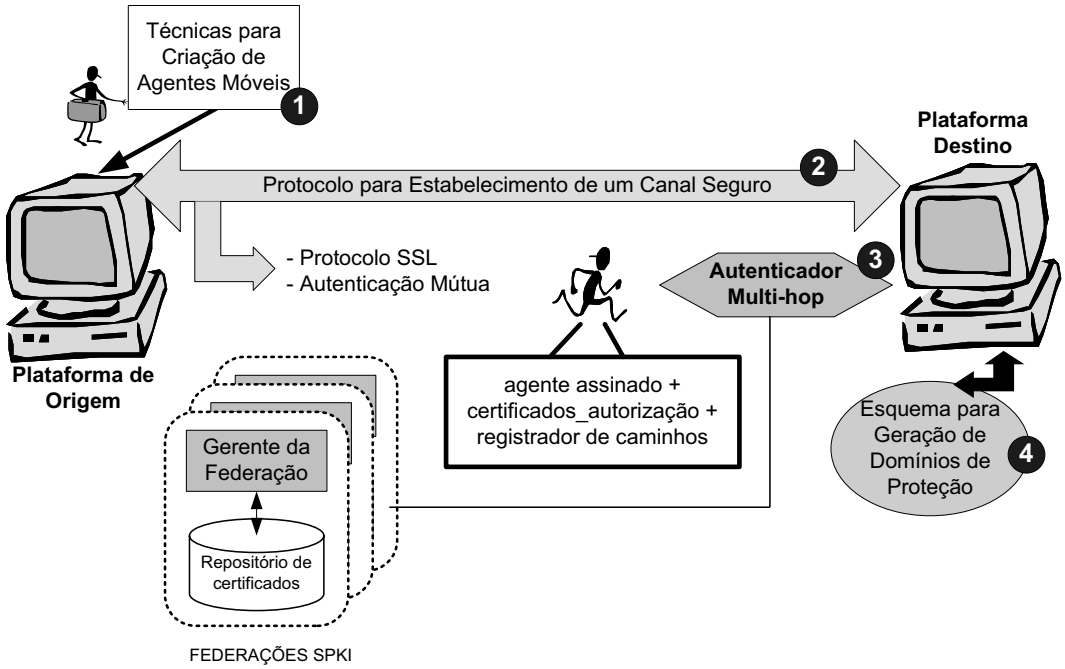


Figura 5.3: $MASS_{ap}$:Esquema de Segurança para Proteção das Plataformas de Agentes

Nas seções a seguir, serão analisados alguns aspectos referentes aos mecanismos que constam no esquema proposto.

5.3.1 Criação de Agentes Móveis

Durante o processo de criação de um agente móvel, o proprietário, na qualidade da autoridade que o agente representa, fornece ao agente um conjunto de cadeias de autorização SPKI/SDSI definindo assim os atributos de privilégios concedidos ao agente. Considerando o estudo de caso dos agentes de viagem (ver seção 3.4), um exemplo de cadeia de autorização em que os atributos de privilégios são expressos através de papéis (*roles*) é apresentado na Figura 5.4. Neste exemplo, a plataforma do Hotel A emite um certificado de autorização, concedendo papéis à plataforma da Companhia Aérea 1, permitindo ainda que este seja delegado (*Cert_Aut 1*). Desta forma, os agentes criados pela plataforma da Companhia Aérea 1 podem exercer os papéis de mensageiro, pesquisador e/ou negociador na plataforma do Hotel A. Para que alguns dos privilégios concedidos em *Cert_Aut 1* sejam delegados para a Agência de Viagens X, a plataforma da Companhia Aérea 1 emite um certificado de autorização (*Cert_Aut 2*), concedendo o papel desejado à plataforma da Agência de Viagens X e ainda fornece o certificado *Cert_Aut 1* para que a cadeia de autorização esteja completa. Quando o agente móvel é criado, este recebe de sua plataforma de origem (Agência de Viagens X) a cadeia de autorização necessária para fazer pesquisas de preços no Hotel A.

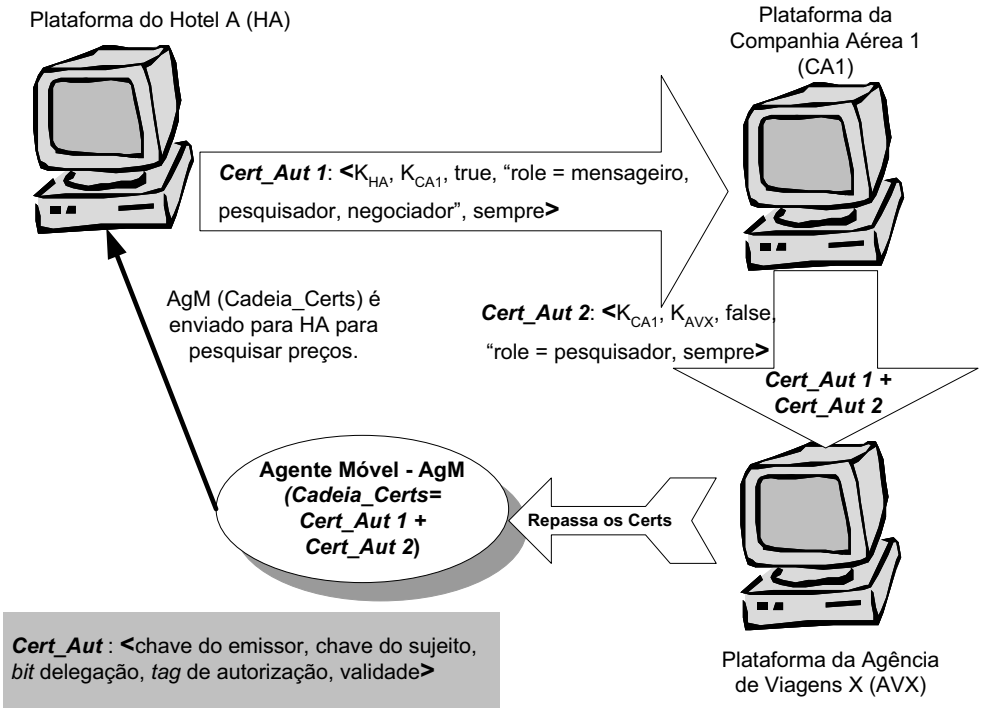


Figura 5.4: Exemplo de Delegação de Certificados de Autorização

É importante ressaltar que essas cadeias iniciais de autorização fornecidas podem não ser suficientes para garantir ao agente o acesso a certos recursos em algumas plataformas. O esquema proposto possibilita que novos certificados possam ser fornecidos ao agente durante suas visitas às plataformas de agentes móveis. Por exemplo, considerando o estudo de caso acima, ao visitar a plataforma de agentes da Companhia Aérea 1, o agente móvel pesquisa-

dor é informado que o Hotel B possui um convênio (relação de confiança) com a Companhia Aérea 1 e que este oferece descontos em suas diárias. Como o agente não possui uma cadeia de autorização que o permita ser recebido na plataforma de agentes deste hotel para realizar uma pesquisa de preços, a própria companhia aérea pode fornecer a este a cadeia de autorização necessária para ser recebido na plataforma do Hotel B. Porém, em alguns casos, estas cadeias de autorização recolhidas durante as visitas dos agentes móveis podem ainda não ser suficientes. A seção 5.3.5 descreve uma proposta, baseada nas federações SPKI, para busca e negociação de cadeias de autorização para os agentes móveis.

Necessária no processo de autenticação de um agente móvel, uma lista indicando as federações de serviço cujas plataformas membros estão autorizadas para executar o agente deve ser definida e anexada pelo proprietário do agente (objeto **lista de federações**). Ainda para auxiliar o processo de autenticação, o proprietário do agente deve criar o objeto **registrador de caminhos** que irá conter informações sobre as plataformas visitadas por este agente. Na iniciação deste objeto, o proprietário deve adicionar uma entrada indicando a sua identidade e a identidade da próxima plataforma a ser visitada. Tanto a **lista de federações**, como o **registrador de caminhos** serão usados para analisar a história das viagens de um agente e assim auxiliar as plataformas de agentes no estabelecimento da confiança neste agente móvel. A autenticação de agentes móveis será discutida na seção 5.3.3.

O esquema de segurança proposto (*MASS*) suporta mecanismos que visam proteger a integridade do código e do estado dos agentes móveis. Como estes mecanismos serão apresentados somente no Capítulo 6, considera-se aqui que o processo de criação de agentes móveis utiliza-se destes mecanismos para proteger o código do agente, a **lista de federações** (dado somente-leitura) e o **registrador de caminhos**, durante o processo de criação do agente.

5.3.2 Estabelecimento de um Canal Seguro entre Plataformas de Agentes

Segundo a especificação MAF (OMG, 2000), as plataformas de agentes móveis, antes de receberem agentes ou de interagirem com outros agentes ou outras plataformas, devem aplicar mecanismos de autenticação e de autorização para conceder ou restringir acessos a seus recursos. No esquema proposto, antes que a transferência de um agente ocorra, a autenticação mútua entre as plataformas envolvidas deve ser estabelecida, tendo como resultado a criação de um canal seguro na infra-estrutura de comunicação entre as partes autenticadas. A autenticação mútua também deve ser realizada quando uma plataforma solicita uma criação remota de um agente ou quando esta deseja invocar, remotamente, métodos de um agente. No $MASS_{ap}$, um agente móvel é visto como um recurso da plataforma em que este está se executando e por isso deve ser protegido.

De acordo com o modelo SPKI/SDSI, a identificação não é feita através de nomes, mas através de chaves públicas, e o mecanismo de autenticação é a assinatura digital. Assim, na

autenticação mútua de plataformas, para que uma assinatura digital seja verificada, a chave pública e uma requisição assinada, devem ser apresentadas à plataforma-destino. No modelo igualitário do SPKI/SDSI, as chaves públicas para verificação de assinaturas são encontradas em cadeias de autorização apresentadas pelas plataformas emissoras às plataformas destinatárias. Ou seja, a base para a autenticação no SPKI/SDSI são as cadeias de certificados de autorização (Elisson, 1999).

No $MASS_{ap}$, a autenticação mútua é executada durante o estabelecimento do canal seguro entre as plataformas de agentes (requisição *establish_trust*). A Figura 5.5 ilustra a autenticação mútua realizada com o uso de um protocolo *Challenge/Response*, conforme descrito em (NIST, 1997), baseado em certificados SPKI/SDSI dos proprietários (ou administradores) das plataformas envolvidas.

No **primeiro passo** da Figura 5.5, a plataforma-fonte envia uma mensagem assinada contendo uma requisição (*establish_trust*) e um *nonce* (*noncePF*), sem qualquer certificado SPKI/SDSI. De posse da requisição, a plataforma-destino monta um desafio assinado (*challenge*) e o envia para a plataforma-fonte para que esta prove a posse dos direitos de acesso para executar a operação (**passo 2**). O desafio é composto pela ACL (*access control list*) que protege o recurso requisitado (execução da operação *establish_trust*), pelo *noncePF* e por um *nonce* gerado pela plataforma-destino (*noncePD*). No passo 3, utilizando a chave pública da plataforma-destino[2], a plataforma-fonte verifica a assinatura do desafio para confirmar a autenticidade da plataforma-destino. Em caso de sucesso, na seqüência, a plataforma-fonte busca a cadeia de autorização que lhe dá o direito de estabelecer um canal seguro com a plataforma-destino para compor a resposta ao desafio e a envia para a plataforma-destino (**passo 3**). A resposta (*response*) é formada pela requisição e pelo *noncePD*, ambos assinados pela plataforma-fonte, mais a cadeia de autorização (*cadeiaCerts*). A partir da cadeia de certificados, a plataforma-destino obtém a chave pública da plataforma-fonte (última chave da cadeia de autorização) e então verifica a autenticidade da plataforma-fonte, terminando assim o processo de autenticação mútua (**passo 4**). Os suportes necessários para a implementação do protocolo de autenticação mútua são providos pelo objeto *SPKI/SDSIResolver* disponíveis em todas as plataformas. E é ainda através deste objeto, que a infra-estrutura SPKI/SDSI se torna disponível em cada plataforma.

É importante frisar que o processo de autenticação mútua das plataformas termina com o estabelecimento de um canal seguro. Este canal permanecerá ativo, sendo usado para a emissão de agentes entre as duas plataformas sem a necessidade da repetição dos passos da Figura 5.5 a cada envio de agentes.

Para o estabelecimento do canal seguro, uma tecnologia de segurança subjacente é usada para garantir a confidencialidade e a integridade das comunicações entre as plataformas de

[2]A plataforma-fonte obtém a chave publica a partir do certifi cado de nome da plataforma-destino que esta possui. Caso não tenha este certifi cado, esta pode ir busca-lo nas federações SPKI/SDSI.

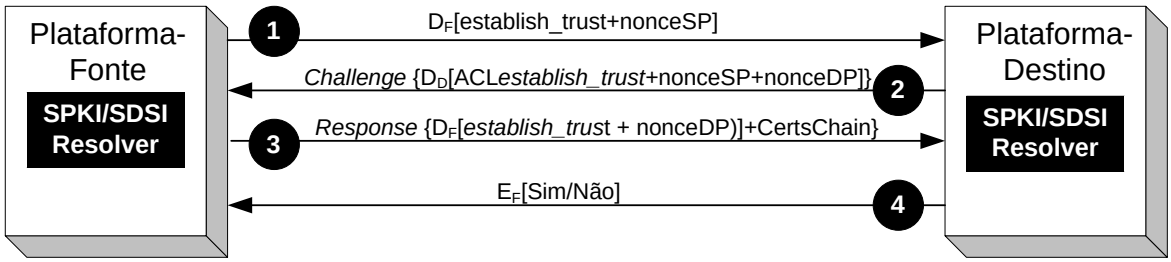


Figura 5.5: Protocolo para Autenticação Mútua de Plataformas

agentes (incluindo o envio de agentes). Dentre as várias tecnologias de segurança em sistemas distribuídos, o SSL (*Secure Socket Layer*) (Freier *et al.*, 1996), desenvolvido pela Netscape, se destaca por ser um protocolo criptográfico amplamente utilizado em aplicações na Internet[3]. Devido a isto, o protocolo SSL foi a tecnologia de segurança escolhida no protótipo do *MASS*.

Quando um canal seguro é estabelecido, a plataforma-fonte envia o agente juntamente com os seus certificados de autorização SPKI/SDSI para que o agente seja autenticado e para que o domínio de proteção apropriado seja criado na plataforma-destino.

5.3.3 Autenticação de Agentes Móveis

Após a autenticação mútua das plataformas, mas antes de iniciar uma *thread* para um agente, a plataforma-destino deve executar o processo de autenticação do agente móvel recebido. Para autenticação de agentes móveis, o mecanismo proposto para autenticação das plataformas de agentes não pode ser usado devido à limitação que impede os agentes de carregarem consigo suas chaves privadas. Neste caso, a especificação MAF (OMG, 2000) sugere o uso de algoritmos, chamados autenticadores, para a verificação da autenticidade dos agentes. Um autenticador usa de informações como a autenticidade da plataforma-fonte, as autoridades do agente e da plataforma de agentes e, possivelmente, informações sobre quais autoridades são confiáveis para autenticar um agente. Porém, esta especificação não trata da autenticação de agentes *multi-hop*. Como contribuição deste trabalho, propõe-se a criação de um **autenticador *multi-hop*** que, baseado na autenticidade do proprietário do agente, na autenticidade das plataformas que o agente visitou e ainda nas listas de federações definidas pelo proprietário do agente, estabeleça a confiança no agente. Como um agente móvel pode se tornar malicioso em virtude do seu estado ter sido corrompido por plataformas visitadas anteriormente (Farmer *et al.*, 1996b), o **autenticador *multi-hop*** também é usado na verificação da integridade do estado do agente e a confiança no agente depende do resultado desta verificação[4].

[3] O SSL é um protocolo de propósito geral adequado para proteger conexões em sistemas distribuídos, prover autenticação, confi dencialidade e integridade para comunicações através de conexões TCP/IP.

[4] Os mecanismos propostos com o objetivo de proteger o agente serão descritos no próximo capítulo.

Considerando o **autenticador** ilustrado na Figura 5.6, uma plataforma, ao receber um agente móvel, deve, primeiramente, através da verificação da assinatura do código do agente, comprovar que este permanece íntegro e confirmar a sua associação a um principal, seu proprietário (**passo 1**). Para agentes *one-hop*, a técnica proposta no passo 1 é suficiente para estabelecer a confiança no agente e autenticá-lo facilmente, porém, para agentes *multi-hop*, esta técnica não é suficiente. Visando analisar o percurso de viagens do agente, a plataforma de agentes de destino deve ainda analisar o objeto **registrador de caminhos** (**passo 2**). Este objeto é criado, iniciado e anexado ao agente pelo seu proprietário com o objetivo de identificar todas as plataformas que este agente percorreu e, com base nesta informação e na lista de federações, ajudar uma plataforma a estabelecer a confiança em um agente. Para isto, cada plataforma visitada deve adicionar no objeto **registrador de caminhos** uma entrada contendo sua identidade e a identidade da próxima plataforma a ser visitada, formando assim a história dos caminhos percorridos pelo agente. As plataformas visitadas deverão estar associadas às federações autorizadas a receber o agente, identificadas na **lista de federações** que o agente carrega consigo.

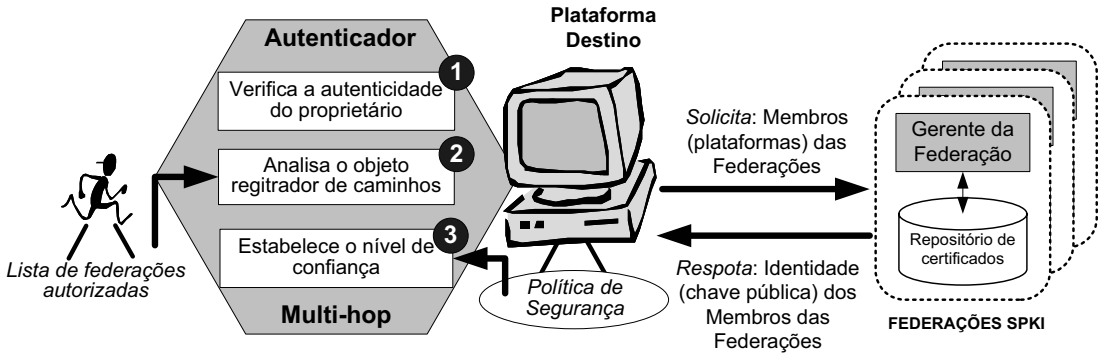


Figura 5.6: Autenticação de Agentes Móveis

Para execução dos **passos 2 e 3** da Figura 5.6, a plataforma deve definir como o objeto **registrador de caminhos** do agente é analisado e como o nível de confiança é estabelecido. No **passo 2**, a plataforma revisa a lista das plataformas visitadas: (1) analisando a consistência do registrador de caminhos e (2) verificando se estas estão associadas a alguma federação indicada na lista de federações[5]. Um inconveniente é que a verificação do caminho tem um custo que cresce à medida que o histórico do caminho aumenta. A Figura 5.6 explicita a solicitação da lista dos membros das federações autorizadas (identificadas na lista de federações). Este passo pode adicionar custo ao desempenho de algumas aplicações. Neste sentido, definiu-se que as listas enviadas pelas federações podem ser armazenadas por um dado período de tempo em uma plataforma, eliminando os custos de comunicação a cada recebimento de agente na plataforma. Após a expiração deste tempo, estas listas precisam ser atualizadas.

A partir dos resultados dos passos anteriores e da política de segurança definida pelo

[5]Para uma correta análise do histórico das viagens de um agente, o objeto **registrador de caminhos** deve ser protegido. Como este objeto faz parte do estado do agente móvel, a sua proteção será descrita no próximo capítulo.

administrador da plataforma para o estabelecimento da confiança em um agente, no **passo 3** é possível definir o nível de confiança nas plataformas visitadas pelo agente e o nível de risco em aceitar o agente

5.3.4 Procedimentos para Geração do Domínio de Proteção

Após estabelecida a confiança no agente (resultado dos procedimentos anteriores) e com base nas cadeias de autorização SPKI/SDSI do agente, os domínios de proteção em que estes serão executados e quais permissões serão atribuídas a estes domínios, precisam ser definidos. As cadeias de autorizações que o agente carrega consigo e que representam suas credenciais (ou atributos de privilégios) necessitam ser verificadas pelo guarda do serviço (guarda da plataforma) para que, em seguida, o conjunto de permissões seja definido e os domínios de proteção do agente sejam gerados (ver Figura 5.7). No $MASS_{ap}$, teve-se a preocupação de desacoplar os atributos de privilégios concedidos aos agentes (cadeias de autorização) dos atributos requeridos ou necessários (atributos de controle ou políticas) para acessar os recursos protegidos da plataforma, proporcionando, assim, um controle de acesso mais flexível, dinâmico e adequado para sistemas de larga escala.

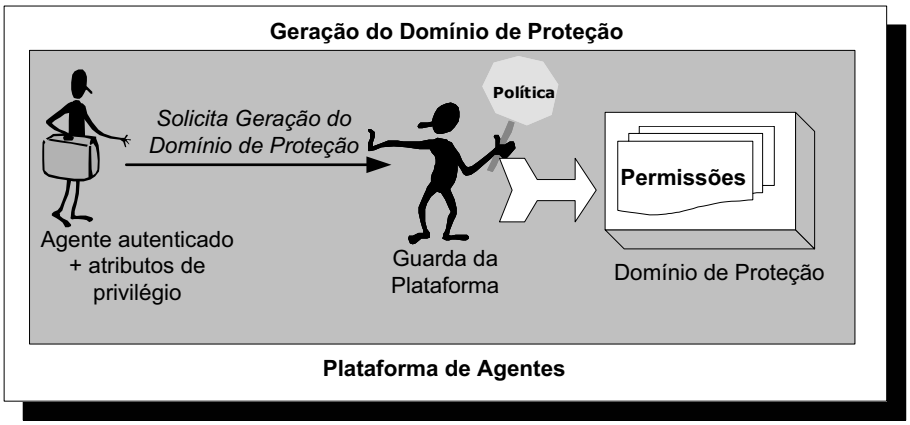


Figura 5.7: Geração do Domínios de Proteção para Execução de Agentes Móveis

A plataforma Java, além de ser considerada um padrão de fato para programação de aplicações distribuídas, possui diversas características e propriedades que a tornam uma boa linguagem para programação de agentes móveis, tais como: interpretação segura de código, portabilidade, programação *multithread*, serialização de objetos, reflexão estrutural, suporte para programação distribuída, carregamento dinâmico de código e assinatura de código (Lange, 1998a). Diante destas características, a linguagem Java e o seu modelo de segurança serviram de base para a concepção do esquema proposto. Embora a plataforma Java seja rica em boas características, esta possui limitações em seu modelo de controle de acesso que é estático (pouco flexível) e centralizado. Como contribuição deste trabalho, os procedimentos para geração de domínios de proteção, conforme ilustrado na Figura 5.8 e descritos a seguir, contornam as limitações do modelo do Java 2.

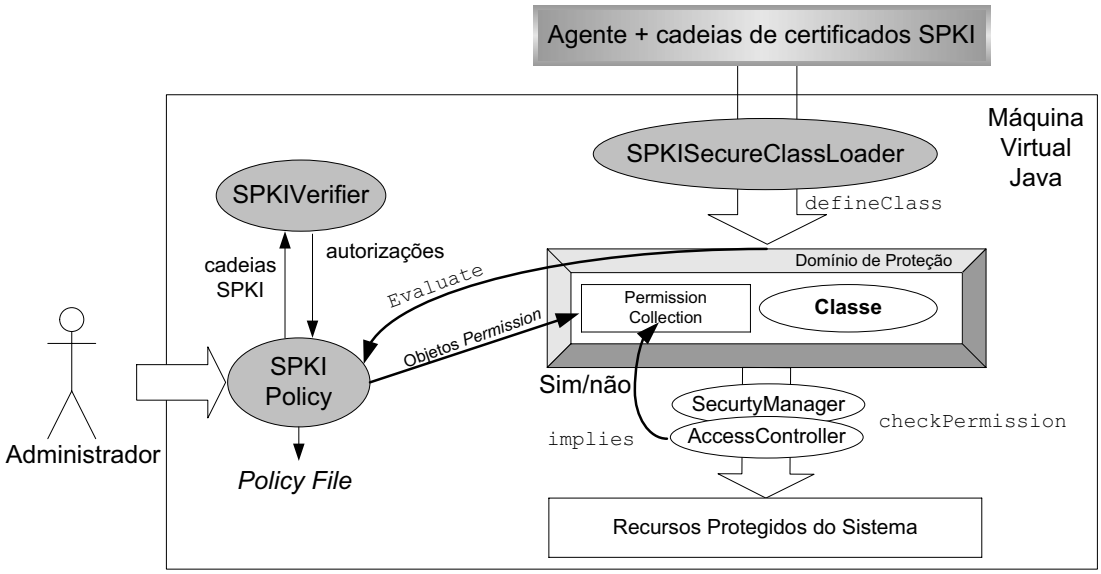


Figura 5.8: Dinâmica para Geração dos Domínios de Proteção Java

Algumas extensões são necessárias no modelo de segurança do Java 2 para a geração do domínio de proteção onde um agente irá se executar (domínio de aplicação). Estas extensões, representadas nos objetos em cinza da Figura 5.8, são: **SPKISecureClassLoader**, necessário para que as cadeias de autorização SPKI possam ser extraídas do agente e para que o domínio de proteção para a *thread* seja criado; **SPKIPolicy**, objeto que representa a política SPKI e que define, a partir dos atributos de privilégios do agente, quais permissões Java serão associadas ao domínio de proteção; e o **SPKIVerifier**, necessário para a checagem dos certificados SPKI.

Seguindo a dinâmica da Figura 5.8, o administrador da plataforma descreve a política de segurança da plataforma de agentes mapeando a autorização concedida, a partir das cadeias SPKI/SDSI em permissões Java, definindo para isto um arquivo de configuração de política (*policy file*). É importante destacar que no esquema proposto este arquivo não é uma ACL que associa todos os objetos do sistema (permissões Java) a sujeitos (proprietários dos agentes), como no modelo de controle de acesso do Java, mas sim um arquivo que associa os objetos do sistema a atributos de privilégios. Quando um agente é recebido na plataforma, seus atributos de privilégios (as cadeias de certificados) são repassadas pelo **SPKISecureClassLoader** para o objeto **SPKIPolicy** que os interpreta. De forma a checar as autorizações possuídas pelo agente, o objeto **SPKIPolicy** repassa as cadeias de autorização apresentadas pelo agente ao objeto **SPKIVerifier**. Quando as autorizações SPKI são mapeadas em permissões Java, o suporte Java gera o domínio de proteção correspondente para a execução da *thread* que se ocupará do agente. As permissões do Java ficam disponíveis no objeto **PermisssionCollection**.

Se a *thread* (agente) faz uma solicitação de acesso a um recurso fora do seu domínio de aplicação, ou seja, em um domínio de sistema, o controlador de acesso (classe **AccessController**) é acionado para verificar se o acesso deve ser permitido. Este controlador de

acesso deve verificar no domínio de proteção criado se o solicitante tem o objeto **Permission** correspondente na sua coleção de permissões. Em caso afirmativo, a *thread* pode trocar de domínio, entrando no domínio de sistema.

5.3.5 Procedimentos para Busca de Novas Cadeias de Autorização no $MASS_{ap}$

A especificação do SPKI/SDSI (Elisson, 1999) determina que o cliente (agente) deve ser o responsável por encontrar os certificados necessários que o ligue a um dado serviço (plataforma de agentes). Ou seja, o próprio agente móvel é quem deve procurar pelos certificados ou cadeias de autorização que este venha a precisar. Para isto, o esquema proposto, usufruindo da infra-estrutura das federações SPKI (Santin, 2004), permite que a plataforma de origem do agente móvel percorra as federações em busca dos certificados de autorização desejados. Quando estes forem encontrados, deve-se iniciar o processo de negociação com o possuidor do privilégio.

Em Santin (2004), o autor apresenta uma heurística para percorrer uma teia de federações visando a formação de novas cadeias. A implementação da estratégia proposta pelo autor em um agente móvel não se mostra uma solução adequada já que as pesquisas nas federações podem ocorrer em paralelo e devido à complexa missão que este agente precisaria desempenhar. Devido a isto, definiu-se que quando um agente móvel constata que não possui as cadeias de autorização necessárias para executar sua missão em uma dada plataforma, este deve retornar para a sua plataforma de origem, para que esta, se assim for desejado, procure na teia de federações SPKI as cadeias de autorização necessárias. A Figura 5.9 exemplifica os procedimentos necessários para busca e formação de novas cadeias de autorização.

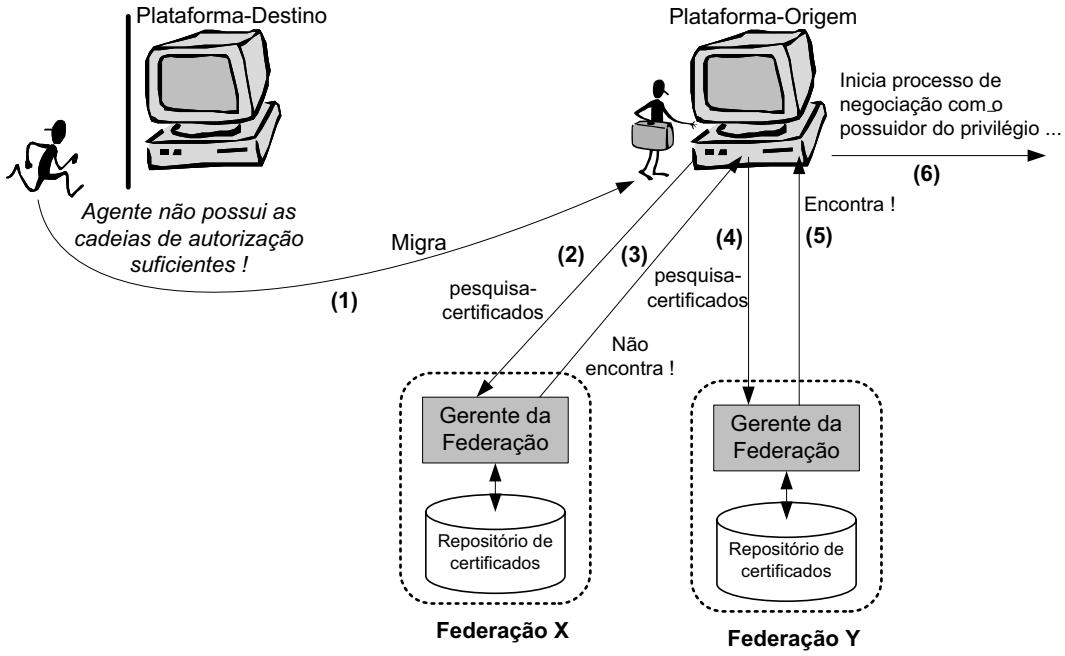


Figura 5.9: Agente Móvel em Busca de Certificados de Autorização em uma Federação SPKI

5.3.6 Comparação com os Trabalhos Relacionados

A maioria das plataformas de agentes móveis, baseadas na linguagem Java, utiliza o modelo de segurança do próprio Java, em especial o do Java 2, para implementar parte de seus mecanismos de segurança. Entre estas, podem ser citadas as plataformas comerciais Aglets da IBM, o Grasshopper da GMDFokus e as plataformas acadêmicas SOMA e Ajanta, desenvolvidas na Universidade de Bologna e na Universidade de Minesota, respectivamente.

Estas plataformas de agentes móveis estendem o gerenciador de segurança do Java (Security Manager) para fornecer uma solução mais flexível e adequada para as plataformas de agentes e para implementar domínios de proteção que isolam os agentes móveis, prevenindo que ataques maliciosos dos mesmos ocorram. Em algumas plataformas, a diferença entre os esquemas de autorização está na informação usada para determinar o conjunto de direitos de acesso de um agente. As plataformas Aglets e Ajanta usam somente a identidade do proprietário do agente. A plataforma Grasshopper faz uso de políticas de controle de acesso baseadas na identidade do proprietário ou no nome do seu grupo (*group membership*). Na plataforma SOMA, os principais são associados com papéis (*roles*) que identificam as operações permitidas sobre os recursos do sistema. Todos esses casos se mostram estáticos ou inadequados para sistemas de larga escala, principalmente, devido às limitações do modelo de controle de acesso do Java 2.

Estas limitações se devem ao fato de que, ao invés de seguir a natureza distribuída do seu modelo de execução, o modelo de segurança do Java 2 está fundamentado em um esquema de autorização centralizado. Essa centralização refere-se ao fato de que todo o controle de acesso é executado a partir de um arquivo único de configuração que define toda a política de segurança de uma máquina. Portanto, tem-se uma única *ACL* relacionando todos os sujeitos e objetos do sistema. Durante a sua execução, cada código é rotulado como pertencente a um ou mais domínios de proteção. Cada domínio possui um conjunto de permissões associadas a partir de um arquivo de configuração de política. Este arquivo define portanto um mapeamento estático entre cada código móvel suportado e as permissões concedidas ao mesmo para a sua execução em um ambiente local. Segundo Molva e Roudier (2000), a confiança das decisões de controle de acesso sobre o arquivo de configuração local resulta em duas grandes limitações:

- cada componente remoto, que pode ser autorizado para acessar recursos locais de uma máquina virtual, deve ser identificado de antemão e registrado no arquivo de configuração de política;
- cada programador de código móvel deve levar em conta, ainda na fase de projeto, as restrições de controle de acesso de todos os possíveis ambientes alvos de execução no sistema distribuído.

Além de numerosas dificuldades práticas em termos de programação, as limitações resultantes da confiança das decisões de controle de acesso sobre o arquivo único de configuração local impedem o desenvolvimento de um ambiente distribuído e dinâmico, exigindo uma definição estática de todos os componentes distribuídos e seus atributos de segurança de uma só vez.

Em comparação com o modelo estático do Java 2 e com as plataformas de agentes móveis, citadas acima, o esquema proposto neste trabalho, apesar de usar algumas funcionalidades do modelo de segurança Java, ao fazer uso de certificados de autorização SPKI, desacopla os atributos de privilégios (credenciais) dos atributos de controle (políticas). Isto significa que, embora se possa continuar definindo estaticamente a política de segurança no arquivo de configuração, o mecanismo proposto ganha em flexibilidade com as possíveis delegações oferecidas pelos certificados SPKI e por este arquivo associar os objetos do sistema (permissões Java) a atributos de privilégios concedidos ao agente e não a identidade dos proprietários dos agentes, como no caso do modelo do Java 2. Ou seja, os domínios são definidos dinamicamente, conforme o agente vá agregando certificados delegados às suas credenciais em seus itinerários.

Para se implantar um esquema efetivo de autorização em plataformas de agentes móveis, a autenticação dos agentes se torna imprescindível. As plataformas Aglets e Grasshopper não possuem mecanismos para autenticação de agentes móveis. Na plataforma SOMA, os agentes são autenticados com base em uma série de informações contidas em suas credenciais. São elas: os nomes do domínio e do lugar de origem; o nome da classe que implementa o agente e o nome do usuário responsável pelo agente. Antes da migração, estas informações, o estado inicial do agente e o código do agente são digitalmente assinados pelo usuário que criou o agente. Quando o agente chega em um sítio remoto, suas credenciais são verificadas em relação à sua autenticidade (assinatura do proprietário do agente). A plataforma Ajanta segue a mesma abordagem da plataforma SOMA na qual as informações das credenciais do agente são usadas para autenticá-lo. A autenticidade de um agente é comprovada pela verificação da assinatura das credenciais que este agente carrega consigo. Em ambas, a abordagem de autenticação de agentes móveis não analisa a confiabilidade das plataformas visitadas pelo agente, mas somente a confiabilidade do proprietário do agente.

Um agente móvel, programado para ser não-malicioso, cujo código está assinado pelo seu proprietário, pode se tornar malicioso em virtude do seu estado ter sido corrompido por plataformas visitadas (Farmer *et al.*, 1996b). Para evitar que permissões sejam concedidas a agentes cujo estado tenha sido corrompido, o processo de autenticação de agentes móveis deve verificar a integridade do estado dos agentes, tanto dos dados mutáveis quanto dos imutáveis. As plataformas SOMA e Ajanta possuem técnicas para proteção do estado do agente. Estas técnicas serão apresentadas e comparadas com as propostas no *MASS* no próximo capítulo. Porém, é importante destacar que nestas plataformas a verificação da integridade do agente não está associada ao processo de autenticação de agentes, e com isso, a verificação ocorre depois dos domínios de proteção já terem sido criados e as permissões concedidas.

No processo de autenticação de agentes móveis do $MASS_{ap}$, conforme descrito na seção 5.3.3, a informação usada para determinar o conjunto de direitos de acesso de um agente e gerar o seu domínio de proteção não está baseada somente na identidade do proprietário do agente, mas nas cadeias de autorização que este agente carrega consigo, na integridade deste agente e também no grau de confiança das plataformas visitadas. Para garantir a integridade dos agentes em trânsito, o $MASS_{ap}$ fornece um canal seguro para o envio de agentes móveis. Além disso, o autenticador *multi-hop* proposto é responsável por verificar a integridade do agente antes de estabelecer o nível de confiança neste mesmo agente. Todo o processo de identificação tanto do agente (seu proprietário) quanto das plataformas visitadas não está baseado em nomes, mas sim em chaves públicas, o que facilita a gestão de nomes em sistemas de larga escala.

Visando aprimorar o controle de acesso do Java 2, dois trabalhos relacionados propõem a utilização de certificados SPKI/SDSI. O primeiro trabalho proposto por Nikander e Partanen (1999) usa certificados de autorização SPKI para delegar permissões Java que descrevem diretamente as possíveis permissões associadas a um domínio de proteção. Nesta proposta, a *tag* de autorização do certificado SPKI foi estendida para expressar as permissões Java. Esta solução traz a desvantagem de usar certificados SPKI modificados. O segundo trabalho (Molva e Roudier, 2000) propõe dois aprimoramentos ao modelo de controle de acesso do Java 2: associação de informação de controle de acesso com cada código móvel (applet) na forma de atributos e introdução de elementos intermediários no esquema de controle de acesso para auxiliar a configuração da informação de controle de acesso em ambientes dinâmicos. Nesta proposta, o mecanismo de grupo do SPKI/SDSI, implementado através de certificados de nomes, é que possibilita esses aprimoramentos. Este trabalho não detalha como implementar esta proposta e como combiná-la com o modelo de segurança atual do Java 2.

Os dois trabalhos comentados acima não estão diretamente voltados para proteção de agentes móveis, mas sim para *applets* Java. Estes não tratam a autenticação mútua entre as plataformas fonte e destino e não analisam o histórico das plataformas visitadas para estabelecer a confiança no código móvel. Somente a primeira proposta possui as características de flexibilidade semelhante ao esquema proposto em que os domínios de proteção são formados segundo os certificados delegados ao agente na sua criação e em seu itinerário. Nikander e Partanen (1999) propõem que a busca para formação de novas cadeias de autorização deve ser de responsabilidade do servidor. Porém, como mencionado na seção 5.3.5, isto não está de acordo com o modelo do SPKI que define que é o cliente (agente ou sua plataforma de origem) deve ser o responsável por encontrar os certificados necessários que o ligue a um dado serviço (plataforma de agentes). Os procedimentos do $MASS_{ap}$ para busca e negociação de novas cadeias de autorização estão em conformidade com o modelo do SPKI.

5.4 Conclusões do Capítulo

As limitações das técnicas apontadas nas seções 5.2 e 5.3.6 comprovam que as questões de segurança relativas a proteção das plataformas de agentes móveis ainda não apresentam resultados satisfatórios que garantam a segurança para as plataformas. Tais limitações são ainda maiores quando se leva em conta aplicações em sistemas abertos que anseiam por escalabilidade, portabilidade e interoperabilidade, já que estas aplicações envolvem vários domínios administrativos e tecnologias heterogêneas. Nesse contexto, as soluções para nomes e segurança devem ser sempre flexíveis, escaláveis e abertas, permitindo que sistemas de grandes dimensões possam evoluir durante a sua dinâmica.

Foi a constatação dessas limitações e a preocupação com estes aspectos específicos da segurança em aplicações de sistemas distribuídos de larga escala que motivaram o esquema de segurança proposto que visa prevenir ataques de agentes móveis contra plataformas, definindo um procedimento que envolve técnicas de prevenção e detecção: o estabelecimento de um canal seguro, a autenticação de agentes móveis *multi-hop* e a geração do conjunto de permissões que será associado aos domínios de proteção criados para o agente. Este esquema está baseado em um controle descentralizado de autorização e autenticação que se mostra adequado para sistemas de larga escala, ao se valer de certificados de autorização SPKI/SDSI. O mecanismo de delegação dos certificados de autorização SPKI possibilita um desacoplamento dos atributos de privilégio do agente dos atributos de controle (políticas), resultando assim em um esquema mais flexível para geração de domínios de proteção, quando comparado com os trabalhos correlatos.

Do ponto de vista da proteção das plataformas de agentes e do canal de comunicação, o esquema de segurança proposto ($MASS_{ap}$) atende os seguintes objetivos de segurança:

- previne a revelação não autorizada do código e do estado de um agente quando este estiver sendo enviado por uma infra-estrutura de comunicação;
- detecta a violação da integridade de um agente móvel ocorrida durante o envio de um agente por uma infra-estrutura de comunicação;
- previne que agentes sejam enviados e recebidos por plataformas não autenticadas;
- registra o caminho percorrido por um agente móvel— as plataformas visitadas por este — minimizando o problema da não repudiação;
- previne o acesso não autorizado de agentes móveis aos recursos locais de uma plataforma de agentes, incluindo os agentes móveis que estejam se executando nesta plataforma.

No próximo capítulo, o problema da proteção de agentes móveis será discutido e um esquema de segurança que visa minimizar os riscos a que esses agentes estão suscetíveis,

quando visitam plataformas maliciosas, será proposto e comparado com os trabalhos relacionados.

Capítulo 6

Proteção dos Agentes Móveis contra Plataformas Maliciosas

6.1 Introdução

Em sistemas baseados em agentes móveis *multi-hop*, tanto as plataformas de agentes temem que os agentes sejam capazes de violar uma plataforma, como os usuários de agentes móveis temem que estes sejam violados quando visitam plataformas maliciosas. Os agentes móveis são executados nos domínios de segurança da plataforma que fornece o ambiente computacional necessário para sua execução. Para executar um agente, uma plataforma deve acessar o código do agente e o seu estado de execução. Os ataques de plataformas maliciosas contra os agentes são os problemas de segurança mais difíceis de serem contornados e ainda encontram-se sem solução adequada. Devido a isto, são poucas as plataformas de agentes móveis comerciais e acadêmicas que focam o problema da segurança dos agentes móveis. As seguintes ameaças são identificadas neste cenário (Jansen e Karygiannis, 1999): mascaramento, negação de serviço, intromissão ou *eavesdropping* e modificação não-autorizada.

Enquanto os mecanismos direcionados à proteção da plataforma são uma evolução direta dos mecanismos tradicionais que enfatizam medidas de prevenção ativa, mecanismos direcionados à proteção dos agentes correspondem normalmente a medidas de detecção. Isto ocorre devido ao fato de um agente ser completamente suscetível à plataforma e de ser difícil evitar a ocorrência de comportamentos maliciosos. Em Chess (1998), há a afirmação de que uma plataforma que executa um dado agente tem total controle sobre as ações desse programa. Por outro lado, em Sander e Tschudin (1998); Loureiro e Molva (1999), estes autores mostram ser possível obter uma execução segura de funções cifradas sobre sítios não confiáveis. Entretanto, os resultados obtidos nos últimos trabalhos mencionados apenas são aplicáveis a um conjunto limitado de funções (mais detalhes na seção 6.2.7).

Outro problema neste cenário esbarra na falta de habilidade de estender efetivamente o ambiente confiável de uma plataforma de origem para outras plataformas. Um usuário

pode assinar digitalmente um agente sobre uma plataforma origem antes deste se mover para uma segunda plataforma. A segunda plataforma, recebendo o agente, pode depender da assinatura para verificar a fonte e a integridade do código e do estado do agente. Em um salto subseqüente do agente para uma terceira plataforma, a assinatura inicial da primeira plataforma permanece válida para o código original e para a informação do estado original, mas não para qualquer estado ou dado gerado na segunda plataforma.

Este capítulo tem por objetivo apresentar uma revisão das principais abordagens e técnicas que visam contornar as ameaças das plataformas maliciosas e descrever ainda o esquema proposto para proteção dos agentes móveis — $MASS_{ma}$. O $MASS_{ma}$ visa diminuir os riscos que os agentes móveis com itinerário livre estão suscetíveis, quando estes visitam plataformas maliciosas, procurando definir protocolos de prevenção e de detecção. Além disso, para atender às necessidades específicas das aplicações, este esquema, baseado em repositórios seguros de dados, pretende ser flexível de modo que o mesmo possa ser especializado através da seleção de um subconjunto de repositórios e protocolos criptográficos. Ainda neste capítulo, o esquema proposto é comparado com os trabalhos relacionados.

6.2 Revisão da Literatura

A segurança de agentes móveis envolve principalmente a **integridade** do agente, para evitar que plataformas alterem o código ou dados que sejam coletados durante as visitas, e a **confidencialidade** do código e do estado do agente, para evitar a violação da propriedade intelectual. Através da técnica de assinatura digital, é possível proteger a integridade do código e dos dados originais de um agente móvel, porém, no caso de agentes *multi-hop*, a integridade dos dados gerados nas plataformas visitadas é difícil de ser garantida.

Para algumas aplicações, esquemas simples de segurança **baseados em confiança** podem mostrar-se adequados. O proprietário de um agente móvel pode restringir o itinerário do mesmo apenas para um conjunto confiável de plataformas previamente conhecidas. Um exemplo é a solução organizacional proposta em Tardo e Valente (1996), em que o sistema de agentes não é aberto e apenas partes confiáveis podem operar nos sítios[1]. A confiança é um relacionamento entre o agente e a plataforma que geralmente não pode ser determinada previamente. Apesar desses esquemas simples baseados em confiança terem o seu valor, esses não suportam itinerários livres, desejáveis em muitas aplicações de agentes.

Uma noção de confiança comumente usada é o **controle social baseado em reputação** (Rasmusson e Jansson, 1996). Mecanismos de controle social, segundo Rasmusson e Jansson (1996), são abordagens de segurança branda (*soft security*), uma vez que estes mecanismos não negam a existência de componentes maliciosos no sistema. Nesta abordagem,

[1]Esta é a abordagem que a General Magic usa em sua plataforma de agentes Telescript.

cada participante do grupo é responsável pela segurança, ou seja, não há uma entidade global ou externa ao grupo que centralize o controle da segurança. Mecanismos que implementam políticas de controle social definem o isolamento dos componentes detectados como sendo maliciosos do convívio de outros componentes corretos de um sistema. Portanto, o controle social estabelece um tipo de comportamento correto que é imposto aos membros de um grupo e cada membro fiscaliza o comportamento dos outros membros. Reputação é a imagem da organização ou entidade, a maneira como ela é vista e reconhecida por aqueles que interagem com ela, direta ou indiretamente (Rasmusson *et al.*, 1997).

Um exemplo de controle social pode ser facilmente construído no cenário do comércio eletrônico. A Internet é um sistema aberto onde componentes e principais podem ter os mais diversos comportamentos e onde não há como se conhecer formalmente as intenções dos principais. Há noções como reputação, estabilidade no mercado e risco estimado que ajudam uma empresa a escolher, por exemplo, com quem esta fará negócios. A partir de um comportamento considerado danoso a uma certa comunidade, pode-se construir um controle social. Entretanto, esta abordagem tem um grave problema: a possibilidade de ataques contra a reputação, em que uma plataforma maliciosa forja provas de uma violação visando comprometer a reputação de uma outra plataforma.

Uma vez que o problema é o comportamento errado do ambiente de execução, contrário a um comportamento que atenda a especificação, algumas abordagens enfatizam o uso de *hardwares* especiais (co-processadores seguros) a prova de ataques, visando construir ambientes confiáveis para computações seguras do agente móvel. Um exemplo é o *Secure Crypto Coprocessor for Workstations* da Citadel. Estas abordagens, entretanto, requerem o uso desses *hardwares* especiais em cada sítio, que é uma condição muito restritiva e muitas vezes cara.

Algumas técnicas de propósito geral para proteger um agente móvel incluem:

- Encapsulamento de Resultado Parcial;
- Contêiner de Dados Somente-Leitura e Vetor de Dados Direcionados;
- Registro de Itinerário Mútuo;
- Registro de Itinerário com replicação e votação;
- Rastro Criptográfico;
- Geração de Chave do Ambiente;
- Computação com Funções Cifradas;
- Ofuscamento de Código.

6.2.1 Encapsulamento de Resultados Parciais

Usada para detectar, em verificações subseqüentes, modificações realizadas por sítios maliciosos, esta abordagem visa encapsular os resultados das ações dos agentes (estados intermediários), praticadas nas plataformas visitadas. Estas verificações podem ocorrer quando o agente retorna a sua plataforma de origem ou ainda a pontos intermediários de verificação. O encapsulamento pode ter diferentes propósitos: fornecer confidencialidade, usando cifragem; fornecer integridade e autenticidade, usando assinatura digital. A informação encapsulada depende dos objetivos do agente, mas, tipicamente, inclui resultados de consultas à base de dados ou *logs* de transações na plataforma. Em geral, há três formas de encapsular resultados parciais (Jansen e Karygiannis, 1999):

- habilitar o agente com meios para encapsular informações;
- depender das habilidades de encapsulamento da plataforma de agentes;
- depender de uma terceira parte confiável.

6.2.1.1 Habilitar o agente com meios para encapsular informações.

Uma característica do encapsulamento controlado pelo agente é que este pode ser aplicado independente da habilidade da plataforma de agente ou da infra-estrutura de suporte. Uma das limitações claramente visíveis é a quantidade e o tamanho das informações a serem recolhidas nas plataformas visitadas. Uma solução chamada *sliding encryption* (Young e Yung, 1997), um modo de operação para criptossistemas de chave pública, permite que pequenas quantidades de dados sejam cifradas e produzam resultados eficazes (economizar espaço ocupado pelos textos cifrados). O cenário para o uso de *sliding encryption* é um cenário em que o agente, usando uma chave pública (que carrega consigo), cifra as informações recolhidas em cada plataforma visitada. Depois, quando o agente retorna para o ponto de origem, a informação é decifrada, usando a chave privada mantida na plataforma. Apesar do propósito da técnica *sliding encryption* ser a confidencialidade, medidas de integridade adicionais podem ser aplicadas antes da cifragem ocorrer (Jansen e Karygiannis, 1999).

Outro método para um agente encapsular informações de resultados é obtido com o uso de Códigos de Autenticação de Resultados Parciais ou PRACs (do inglês, *Partial Result Authentication Codes)* (Yee, 1997), que são similares aos MACs (*Message Authentication Codes*)— *cheksums* que usam criptografia simétrica. No PRAC, a preocupação está em demonstrar a autenticidade de um estado intermediário. A técnica, proposta por Yee, visa garantir a **propriedade de integridade em avanço** (*forward integrity*). Segundo esta propriedade, se um agente móvel visita uma seqüencia de plataformas P = P_1, P_2,..., P_n e a primeira plataforma maliciosa é P_c, então nenhum dos resultados obtidos em plataformas anteriormente visitadas, isto é, P_i com i < c, pode ser forjado. Esta técnica requer que o agente e o

seu originador mantenham ou gerem incrementalmente uma lista de chaves secretas usadas na computação PRAC. Então, uma vez que uma chave é aplicada para encapsular a informação coletada, o agente destrói esta chave antes de se mover para a próxima plataforma, garantindo a integridade em avanço. Desta forma, o originador do agente deve manter uma ou mais chaves secretas para detectar modificações nos resultados parciais coletados.

Para permitir que um PRAC seja verificado publicamente por uma plataforma intermediária, Yee (1997) propõe o uso de um sistema de assinatura digital. Quando um agente é disparado, este carrega consigo uma lista contendo um conjunto de funções de assinatura digital $Sig_i(x)$, bem como outra lista contendo as respectivas funções de verificação $Verif_i(x)$. Em um servidor P_i, o agente utiliza a função $Sig_i(x)$ para gerar o PRAC do resultado coletado e a elimina antes de migrar para P_{i+1}.

A técnica de PRAC tem algumas limitações, conforme descritas pelo próprio autor (Yee, 1997) e também em Karjoth *et al.* (1998). A mais séria limitação ocorre quando uma plataforma maliciosa retém cópias das chaves originais ou das funções de geração das chaves de um agente. Outro problema apontado é que uma entrada de resultados parciais ou séries de entradas encapsuladas entre duas visitas a uma mesma plataforma maliciosa podem ser alteradas ou destruídas sem a possibilidade de detecção pela plataforma de origem. O problema persiste se uma plataforma maliciosa forma um conluio com outra plataforma posteriormente visitada, informando a esta os dados necessários para a computação de PRACs. Devido a estes problemas, esta técnica possui resistência fraca a alteração e destruição de dados.

6.2.1.2 Depender das habilidades de encapsulamento da plataforma agente

Ao invés de depender do agente para encapsular informações, cada plataforma pode ser requerida para encapsular resultados parciais no caminho (Chess *et al.*, 1995). Os seguintes trabalhos seguem esta abordagem de encapsulamento: **Protocolos KAG** (Karjoth *et al.*, 1998), **Contêiner Somente para Inclusão do Ajanta** (Karnik, 1998) e **Protocolo Múltiplo-salto** da plataforma SOMA (Corradi *et al.*, 1999a).

Karjoth *et al.* (1998) propõem uma família de protocolos orientada à plataforma para o encapsulamento de resultados parciais — **protocolos KAG**, que reformula e melhora a técnica PRAC, proposta por Yee (1997). Nos protocolos propostos, um agente móvel visita uma seqüencia de plataformas P = P_1, P_2,..., P_n, sendo que este parte de P_0 (originador) e volta para $P_0 = P_{n+1}$. A abordagem possibilita construir uma cadeia de resultados encapsulados (*hash chainning*) que liga cada entrada de resultado a todas as entradas anteriores e à identidade da próxima plataforma a ser visitada. Cada plataforma assina digitalmente a sua entrada (chamada oferta), que opcionalmente pode estar cifrada, usando a sua chave privada e usa uma função *hash* segura para ligar o resultado da plataforma anterior (oferta anterior) e a identidade da próxima plataforma (protocolos P1 e P2 (Karjoth *et al.*, 1998)). Uma variante desta técnica, que usa MAC ao invés de assinaturas digitais, é também descrita

em Karjoth *et al.* (1998)— protocolos P3 e P4. Baseado na propriedade *forward integrity* de Yee, Karjoth *et al.* (1998) definem a propriedade *strong forward integrity* – **integridade em avanço forte**. Segundo esta proposta, nenhum estado encapsulado, chamado pelos autores de oferta encapsulada, representado por O_k, obtido em uma plataforma P_k, pode ser modificado sem ser detectado. Nesta definição, para garantir uma resistência forte à alteração de alguma oferta, mesmo quando uma plataforma maliciosa for visitada duas vezes ou quando um conluio entre plataformas for constituído, tem-se a restrição de que o último elemento na cadeia de estados encapsulados O_m não pode ser modificado ou excluído.

Avaliando a robustez dos protocolos KAG, Roth (2001) constata que não há motivos para acreditar que um invasor não corromperia O_m, se necessário. Além disso, Roth descreve um ataque que visa violar a integridade de resultados parciais coletados e introduzidos pelas plataformas através dos protocolo P1 e P2 da família KAG. Para isto, uma plataforma maliciosa separa os resultados encapsulados $(O_o, ..., O_i)$ do programa do agente correto, chamado $prog_A$, anexa esses resultados parciais a um novo código malicioso, chamado $prog_M$ e, explorando algumas plataformas como um oráculo, faz com que essas incluam resultados parciais que atendam as especificações do agente malicioso (p. ex., O_j, O_{j+1}, O_{j+2}). Em seguida, esse agente malicioso retorna à plataforma maliciosa que remonta os resultados encapsulados com o agente correto ($prog_A, O_o, ..., O_i, O_j, O_{j+1}, O_{j+2}$). Os resultados parciais que atendem as especificações de um agente malicioso passam a fazer parte do estado de um agente correto sem que isto possa ser detectado. Isto ocorre devido à falta de uma redundância criptográfica que ligue o resultado encapsulado à instância do agente. Roth afirma ainda que todos os protocolos da família KAG possui resistência fraca a destruição total das ofertas encapsuladas (*weak truncation resilience*), já que é possível que isto aconteça quando um plataforma maliciosa é visitada mais de uma vez ou quando um conluio de plataformas é formado.

O mecanismo **Contêiner Somente para Inclusão** (*Append-Only Container*), proposto em Karnik (1998), faz parte da arquitetura de segurança da plataforma Ajanta. Este mecanismo permite que os resultados parciais (dados) obtidos nas plataformas visitadas sejam incluídos de forma segura em um objeto chamado *AppendOnlyContainer*. O objeto *AppendOnlyContainer* contém três vetores que armazenam os objetos incluídos, as respectivas assinaturas digitais e as identidades dos assinantes, além de um *checksum* (vetor de *bytes*) que permite que a plataforma de origem detecte modificações indesejáveis. Caso a plataforma deseje manter segredo acerca de alguns dados sensíveis gerados por esta e que serão carregados pelo agente, esta plataforma pode ainda cifrá-los com a chave pública do proprietário do agente para que somente este tenha acesso aos dados. Quando um agente é criado, a plataforma de origem gera um *nonce* N_o, que deve ser mantido em segredo, e que é usado para calcular o *checksum* inicial. O *checksum* é inicializado pela cifragem do *nonce* com a chave pública do proprietário do agente ($checksum = ENC_o(N_o)$). Dados armazenados neste objeto não podem ser removidos ou modificados sem que o proprietário do agente os detecte, já que antes de incluir um dado, a plataforma deve assiná-lo e o *checksum* sobre os resultados

inseridos no objeto deve ser atualizado. Quando um objeto X é adicionado no contêiner pela plataforma corrente C, o novo valor do *checksum* é calculado da seguinte forma:

$$checksum = ENC_o[checksum + Sig_C(X) + C]$$

Primeiro, a assinatura do objeto X e a identidade do assinante são concatenados ao valor corrente do *checksum*. E então, o valor do checksum é computado pela cifragem deste vetor de *bytes*, usando a chave pública do proprietário do agente[2]. Neste mecanismo, além de o contêiner somente poder ser verificado na plataforma de origem do agente, o uso do checksum não garante a propriedade de integridade em avanço. Ou seja, quando uma violação é detectada, os resultados parciais coletados anteriormente ao encapsulamento do resultado corrompido não podem ser aproveitados.

Roth (2001) apresenta algumas situações de ataque que não podem ser detectadas pelo esquema apresentado por Karnik (1998). Segundo aquele autor, uma plataforma maliciosa que é visitada mais de uma vez por um mesmo agente móvel ou que participa de um conluio com outras plataformas pode remover todos os dados coletados entre as duas visitas e adicionar dados falsos ou pode adicionar ou remover objetos arbitrários armazenados entre as duas visitas sem que isso seja detectado pela plataforma de origem. Visando contornar esses ataques, em Karnik e Tripathi (2001), os autores estendem este mecanismo passando a contar com uma entidade confiável, como o criador/guardião do agente, para manter um registro das inserções dos ítens no contêiner. Cada vez que um ítem é adicionado, um número sequencial seguro (SSN - *Secure Sequence Number*) é atribuído ao ítem pelo guardião do agente ou por alguma outra entidade confiável. Neste novo esquema, o objeto *AppendOnlyContainer* necessita de um novo vetor para armazenar os SSNs. Com esta extensão, este mecanismo passa a depender não só da plataforma para encapsular os resultados parciais, mas também de uma terceira parte confiável. Além disso, o ataque que explora uma plataforma como um oráculo, violando a integridade dos resultados parciais encapsulados, descrito acima para os protocolos da família KAG, também é possível no Contêiner Somente para Inclusão.

No **Protocolo Múltiplo-Salto** (Corradi *et al.*, 1999a) da plataforma SOMA, um agente é composto por três partes: os dados de inicialização e o código, chamado CID, os dados de aplicação, chamado AD, e os dados do protocolo, chamado PD. O CID é a parte imutável do agente protegido pela assinatura do proprietário. A parte AD contém o dado coletado pelo agente em cada salto. A integridade do dado coletado é garantida pela parte PD que registra as informações necessárias para a verificação da integridade. Neste protocolo, cada plataforma deve fornecer uma pequena prova da computação do agente que está armazenada na parte PD do agente. Cada prova é criptograficamente ligada com as computações dos sítios anteriores (estabelecendo uma relação de encadeamento entre as provas) para que seja impossível modificar uma prova intermediária sem modificar também todas as provas posteriores. Quando o agente volta para o sítio do seu proprietário, a integridade da cadeia

[2]O valor anterior do checksum deve ser apagado do estado do agente.

de provas criptográficas é verificada, permitindo ao proprietário detectar qualquer violação de integridade. O protocolo Múltiplo-Salto também é vulnerável a ataques que removem dados coletados quando um sítio malicioso pode ser visitado mais de uma vez pelo mesmo agente, ou se dois sítios maliciosos formam um conluio para atacar um determinado agente, conforme descrito em Roth (2001).

6.2.1.3 Depender de uma terceira parte confiável.

Yee, que propôs a técnica PRAC, observou que a integridade em avanço pode também ser encontrada usando uma terceira parte confiável que executa datação de informação. Yee (1997) levanta a preocupação de que a granularidade dos *timestamps* possam limitar a taxa de viagem máxima de um agente, uma vez que o passo seguinte só pode ser executado no próximo *ticket* do relógio global. É óbvio que esta abordagem necessita de relógios sincronizados.

Além do protocolo Múltiplo-Salto, a plataforma SOMA (Corradi *et al.*, 1999a) possui ainda outra solução para verificar a integridade dos resultados parciais que o agente carrega consigo, que se vale de uma terceira parte confiável, chamada TTP, para validar os dados coletados pelo agente. Nesta solução, um agente sempre carrega um MIC (*Message Integrity Code*) calculado pela terceira parte confiável sobre o dado coletado em um lugar visitado. Para isto, depois de cada visita a um sítio não confiável, o agente deve migrar para o sítio da TTP para que esta verifique o MIC, previamente calculado, visando assegurar que nenhum dado coletado anteriormente tenha sido modificado pela plataforma visitada. Após essa verificação, a TTP recalcula o MIC sobre o MIC anterior e sobre os últimos dados coletados, e envia o agente para o próximo lugar. Esta solução acarreta perdas de desempenho devido à necessidade do agente migrar para uma TTP sempre que este visitar um sítio não confiável. A segurança do protocolo está baseada na suposição de que apenas as TTPs conhecem a função *hash* utilizada para gerar o MIC do estado do agente (prova criptográfica), já que o valor do *hash* não é assinado. Como a segurança de primitivas criptográficas não deve depender de tal suposição, Uto (2003) sugere a utilização de um MAC, ao invés do MIC, cujas chaves criptográficas usadas na assinatura sejam compartilhadas pelas TTPs.

6.2.2 Contêiner de Dados Somente-Leitura e Vetor de Dados Direcionados

Além do mecanismo Contêiner Somente para Inclusão, a plataforma Ajanta oferece ainda dois mecanismos para proteger os agentes contra plataformas maliciosas. Estes mecanismos permitem detectar modificações indesejadas no estado do agente (Karnik, 1998). O primeiro mecanismo, **Contêiner de Dados Somente para Leitura**, permite ao programador, durante a criação do agente, declarar partes do estado do agente como dados somente para leitura. Sobre esses dados, um *hash* de 128 bits é calculado para que então o proprietário do agente

assine com a sua chave privada. O algoritmo DSA (*Digital Signature Algorithm*) é usado para este propósito. O agente carrega consigo esta assinatura permitindo assim que qualquer plataforma possa verificar a proteção dos dados, utilizando a chave pública do proprietário do agente. Conforme descrito em Karnik (1998), o programador do agente deve armazenar os dados a serem protegidos em um vetor de objetos (chamado *objs*) para então criar o objeto *ReadOnlyContainer* que armazenará o vetor e a sua assinatura. Visando prevenir alguns ataques descritos em Roth (2001), e em Karnik e Tripathi (2001), este mecanismo é estendido acrescentando as credencias do agente ao conjunto de dados somente-leitura a serem assinados. As credenciais contêm informações para a autenticação do agente, entre elas, o nome global do agente, o *code base* e as identidades (em formato URN- *Uniform Resource Name*) do proprietário do agente e do seu criador, necessárias para propósitos de controle de acesso. Esta extensão liga as credenciais de um agente ao seu estado somente-leitura.

O segundo mecanismo (chamado de ***TargetedState***) implementa um vetor de dados direcionados em que cada posição tem plataformas específicas como destinatárias, possibilitando uma revelação seletiva do estado do agente . Ou seja, o programador do agente pode especificar que certos objetos carregados pelo agente deverão somente estar visíveis para plataformas pré-definidas. Para isto, o agente, durante sua criação, cifra o dado em questão com a chave pública do destinatário. Esses dados direcionados são ainda assinados pelo proprietário do agente. Assim cada plataforma, antes de decifrar o objeto a ela destinada, pode verificar a assinatura digital do vetor de dados. Roth (2001) descreve um possível ataque que este mecanismo não detecta. Este ataque ocorre devido à falta de uma redundância criptográfica que interliga o conteúdo cifrado à instância do agente. Para concretizar este ataque, uma plataforma maliciosa deve separar a assinatura do proprietário do agente do vetor de dados, copiar o vetor de dados dentro de um outro vetor de um agente criado pela plataforma maliciosa, assinar este novo vetor de dados e enviar para as plataformas destinatárias dos estados. Quando as plataformas destinatárias recebem este agente malicioso, estas inocentemente decifram o dado direcionado a elas e tornam o resultado decifrado disponível para o agente. Para o ataque ser concluído, basta então que este agente retorne para a plataforma maliciosa com os dados decifrados.

6.2.3 Registro de Itinerário Mútuo (*Mutual Itinerary Recording*)

Uma variação interessante do Histórico dos Caminhos é um esquema geral que permite que um itinerário de um agente seja registrado e rastreado por outro agente cooperante e vice-versa (Roth, 1998), em um suporte de acordo mútuo. A idéia geral é executar tarefas críticas como autorização de negociações entre uma plataforma e um agente A em um outro agente cooperante B e dividir ou compartilhar dados referentes a estas negociações entre os agentes (A e B). Um agente cooperante A deve visitar somente plataformas contidas em H_A, e o seu par, o agente B, somente deve visitar plataformas contidas em H_B, onde H_A e H_B são

conjuntos de plataformas não vazios e disjuntos. Assim A e B nunca visitarão a mesma plataforma. Quando o agente migra entre plataformas, este deve repassar informações da última plataforma, da plataforma corrente e da próxima plataforma para o par cooperante, através de um canal autenticado. O par cooperante (A e B) mantém um registro do itinerário e toma ações apropriadas quando inconsistências são observadas. Este esquema está fundamentado sobre a suposição de que são poucas as plataformas maliciosas. Caso um agente A encontre uma plataforma maliciosa, esta não vai participar de um conluio com outras plataformas a serem visitadas pelo seu par cooperante B.

Alguns inconvenientes desta técnica incluem o custo para definição de um canal autenticado e a falta de habilidade do par cooperante para determinar qual das duas plataformas (a que enviou o agente ou a que o recebeu) é responsável por alguma incoerência. Além disso, as suposições adotadas na definição do protocolo são difíceis de serem constatadas de antemão (as plataformas a serem visitadas pelo agente A não participam de conluio com as plataformas definidas no itinerário do agente B).

6.2.4 Registro de Itinerário com Replicação e Votação

Uma plataforma que possui falhas pode apresentar um comportamento malicioso que pode corromper o código ou o estado de um agente. O uso de técnicas de tolerância a faltas pode ajudar a contornar os efeitos de plataformas maliciosas. Uma dessas técnicas que pode assegurar a chegada de um agente móvel de forma segura ao seu destino é a replicação com voto majoritário, conforme proposto por Schneider (1997). A idéia é que, ao invés de uma única cópia de um agente executar uma computação, múltiplas cópias do agente sejam usadas. Embora algumas plataformas maliciosas possam corromper cópias do agente, as réplicas restantes, através de votação, podem permitir o sucesso da computação, desde que a maioria se execute em plataformas corretas. Conforme ilustrado na Figura 6.1, a primeira plataforma é chamada fonte e a última da seqüencia de plataformas visitadas (trajetória) é chamada de plataforma de destino. As demais plataformas pertencem cada uma a um estágio que é replicado para prover a tolerância a falhas.

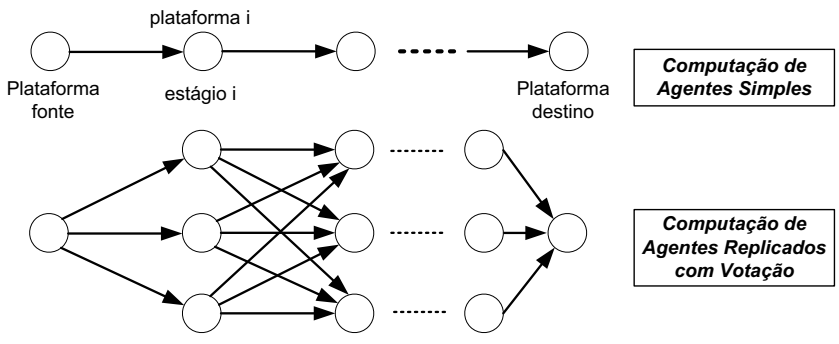


Figura 6.1: Replicação e Votação

Para cada estágio da computação, uma plataforma deve testar os agentes que chegam,

verificando se os mesmos estão intactos e carregam credenciais válidas. A plataforma propaga para o próximo estágio apenas o subconjunto de agentes replicados que esta considera válido. Um dos protocolos usados nesta técnica requer que os agentes transportem as assinaturas acumuladas de todos os nós por onde estes passaram. A abordagem é similar ao Histórico dos Caminhos, mas estende com qualidade de serviço referente a tolerância a faltas. A técnica parece ser apropriada para aplicações especializadas onde agentes podem ser duplicados sem problemas e o processamento geral pode ser estruturado como uma computação *multi-staged*, em que a sobrevivência do agente é a maior preocupação (Jansen e Karygiannis, 1999). Um inconveniente óbvio são os recursos adicionais consumidos pelos agentes replicados.

6.2.5 Rastro Criptográfico

A técnica de Rastros Criptográficos, proposta em Vigna (1997, 1998a), foi concebida para detectar modificações não-autorizadas em um agente através de um registro fiel do comportamento do agente e sua execução sobre cada plataforma de agente. Desta forma, é possível detectar modificações maliciosas nos valores de variáveis e no código do agente, bem como detectar o desvio de fluxo de execução e cobrança indevida de serviços. A técnica requer que cada plataforma envolvida crie e mantenha um registro não repudiável ou o rastro das operações executadas pelo agente enquanto residente nesta plataforma. Um *hash* criptográfico do rastro é gerado, sendo o mesmo composto de uma seqüência de pares <*n*,*s*> onde *n* representa um identificador único de uma declaração e *s* é, chamada pelo autor, de assinatura obtida pela plataforma. Por exemplo, se a instrução *read (x)* que lê uma variável *x* de um terminal obtiver o inteiro *3*, a assinatura associada será *x:=3*. A assinatura é necessária apenas para as instruções que dependem das interações com o ambiente computacional mantido pela plataforma. Para instruções que dependem apenas de valores de variáveis internas, a assinatura não é requerida.

Vigna (1997, 1998a) apresenta alguns protocolos criptográficos, baseados no uso de rastros, para transportar agentes e para associar informações relativas a segurança entre as várias partes envolvidas. Uma terceira parte confiável pode ser usada para conservar a seqüencia dos resumos dos rastros para todo o itinerário do agente. Se algum resultado suspeito ocorrer, os rastros apropriados e os resumos podem ser obtidos e verificados. Para identificar uma plataforma maliciosa, o proprietário do agente deve executar o agente novamente em um simulador e verificar o rastro obtido (histórico da execução do agente).

A abordagem tem inúmeros inconvenientes, sendo os mais óbvios o próprio tamanho e o número de registros, e ainda o fato de que o processo de detecção pode ser ocasionalmente desencadeado, baseado em resultados apenas suspeitos. Além disso, de acordo com as premissas definidas em Vigna (1997), os agentes não podem compartilhar memória e devem possuir só uma *thread* (sem concorrência interna).

6.2.6 Geração de Chaves de Ambiente

Esta técnica descrita em Riordan e Schneier (1998) introduz um esquema que permite que um agente tome ações pré-definidas quando alguma condição do ambiente for verdadeira. A abordagem está centrada sobre a construção de agentes (*clueless agents*) de forma que, encontrando uma condição do ambiente, uma chave é gerada pelo agente. Esta chave gerada será usada para desbloquear algum código executável embutido em uma mensagem cifrada. A condição do ambiente necessária para a construção da chave é protegida por uma função *hash one-way*. A geração de chaves é similar a idéia das chaves *ephemeral* que são aleatoriamente criadas em um determinado tempo de uso e destruídas mais tarde.

A técnica assegura que uma plataforma maliciosa ou um observador do agente não descubra o conteúdo de mensagens trocadas pelos agentes ou responda a essas mensagens lendo o código do agente. Agentes, com dados ou código executável cifrados, podem esconder seus propósitos até que alguma condição do ambiente seja atendida. Como exemplos de dados coletados nos ambientes (condições) para geração de chaves, tem-se em Riordan e Schneier (1998): mensagens ou partes de mensagens de correios eletrônicos ou grupos USENET, nomes de arquivos, nome de domínio DNS e dados temporais. Uma preocupação necessária é que não se deve utilizar um domínio de dados com poucos elementos o que possibilitaria realizar ataques de dicionário (Uto, 2003). O procedimento é um tanto quanto parecido com a forma na qual as senhas são mantidas em sistemas operacionais Unix (p.ex.: como o *hash* de uma senha é armazenado). Em Riordan e Schneier (1998), são propostas três abordagens para geração de chaves criptográficas, a partir de observações do ambiente. Uma limitação desta abordagem é que uma plataforma de agentes, tipicamente, limita a capacidade de um agente para executar códigos criados dinamicamente, já que isto é considerado uma operação não segura (Jansen e Karygiannis, 1999).

6.2.7 Computação com Funções Cifradas

Os argumentos, que suportam a crença de que a confidencialidade da computação dos agentes móveis não pode ser conseguida sem *hardware* resistente a modificações, são considerados errôneos por Sander e Tschudin (1998). Segundo os autores, em muitos casos, soluções criptográficas baseadas em *software* podem proteger os agentes móveis contra sítios maliciosos.

O objetivo da técnica proposta em Sander e Tschudin (1998), também conhecida como Criptografia Móvel, é não deixar em claro partes sensíveis do código. Nesta abordagem, a plataforma de agentes deve executar funções cifradas sem ser capaz de discernir a função original.

A Figura 6.2 ilustra o protocolo não interativo para computação com funções cifradas. Segundo este protocolo, João tem um algoritmo para computar uma função f. Maria tem

uma entrada *x* e quer computar *f(x)*. João não quer que Maria aprenda qualquer coisa sobre *f* e não quer que Maria interaja com ele durante a computação de *f(x)*. Se *f* pode ser cifrado em uma forma que resulte em outra função *E(f)* (passo 1, Figura 6.2), então João pode criar um programa *P(E(f))* (passo 2), que implementa *E(f)* e enviá-lo para Maria, embutido em um agente (passo 3). Maria então executa *P(E(f))* em *x* (passo 4), e retorna o resultado para João (passo 5) que o decifra para obter *f(x)* (passo 6).

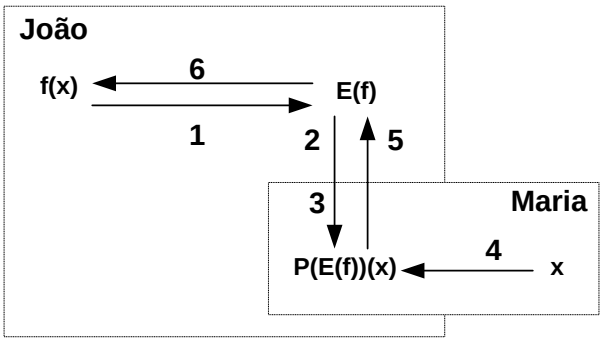


Figura 6.2: Protocolo Não-Interativo para Computação com Funções Cifradas

Pode-se constatar que se *f* for um algoritmo de assinatura com uma chave embutida, o agente teria meios efetivos para assinar informações sem que a plataforma descubra a chave. De forma similar, se *f* for um algoritmo de cifragem com uma chave embutida, o agente também teria meios efetivos para cifrar informações na plataforma.

Embora a idéia seja simples, difícil é encontrar esquemas de ciframento apropriados que possam transformar funções arbitrárias sem alterar os resultados da aplicação das mesmas. Os resultados inicias de Sander e Tschudin parecem promissores segundo Jansen e Karygiannis (1999). Os autores da técnica já conseguiram empregar o protocolo descrito acima para funções polinomiais e racionais usando técnicas de composição de funções e esquemas de ciframento, contendo as propriedades homomórficas necessárias e encontraram uma técnica para esconder uma função simples de assinatura, tornando as assinaturas não detectáveis. Entretanto, este processo de assinatura faz uso de cifragens fáceis de serem invertidas e, na prática, não se conhece um método eficiente para gerar esta transformação. Espera-se que esses resultados formem a base para a descoberta de outras classes de funções. A técnica, apesar de muito poderosa (garante a integridade e confidencialidade de funções), não previne contra a negação de serviço, ataque de mensagem antiga e extração experimental (Jansen e Karygiannis, 1999).

6.2.8 Ofuscamento de Código

Segundo a técnica chamada Segurança de Caixa-preta (*Blackbox Security*), um agente é considerado como tendo a propriedade caixa-preta se em qualquer instante de tempo, o código e os dados de um agente não puderem ser revelados e modificados (Hohl, 1998). Somente a entrada e a saída podem ser observadas. O mecanismo de conversão que gera

um agente com a propriedade caixa-preta usa parâmetros aleatórios de configuração que permitem criar diferentes caixas-pretas de uma mesma especificação (ver figura 6.3).

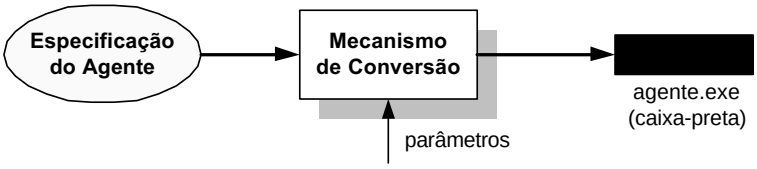


Figura 6.3: Abordagem Caixa-Preta

Se um agente atende totalmente à propriedade caixa-preta definida acima, este é autônomo no sentido de que se um sítio executa o agente, este não pode interferir em sua execução diretamente. O grande problema está em encontrar a propriedade caixa-preta. A abordagem de Computação com Funções Cifradas (ver seção 6.2.7) pode ser classificada dentro da abordagem caixa-preta e é a única técnica que atende os requisitos de "caixa-preta"(Hohl, 1998). Entretanto, esta técnica, no estágio atual, não é aplicável para qualquer tipo de agente. Como alternativa, Hohl (1998) redefine a propriedade caixa-preta de uma forma que difere na declaração sobre quanto tempo a proteção permanece válida. Agora, a proteção não é mais para sempre, mas apenas por um limitado e conhecido intervalo de tempo. Um agente segue a abordagem caixa-preta limitado no tempo se para um certo intervalo de tempo, o código e os dados de um agente não puderem ser revelados. Para tornar o intervalo de proteção explícito, uma data de expiração é anexada à caixa-preta (ver Figura 6.4).

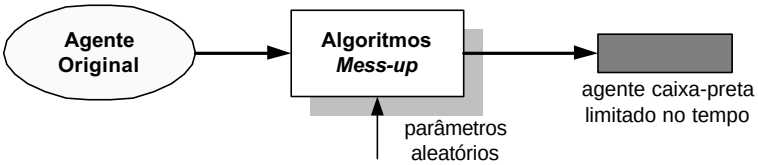


Figura 6.4: Abordagem Caixa-Preta Limitada no Tempo

A idéia chave desta técnica é não permitir que um atacante construa de antemão um modelo mental do agente (engenharia reversa). Um exemplo da abordagem caixa-preta limitada no tempo são as transformações de ofuscamento de código, que criam novas formas para os agentes, tornando difícil o entendimento dos códigos e dos dados dos mesmos. Ou seja, o ofuscamento de código é uma técnica que converte um programa em outro programa com o comportamento similar, mas é muito difícil de ser entendido quando técnicas de engenharia reversa são aplicadas.

Além da técnica caixa-preta limitada no tempo, muitos outros trabalhos seguem a abordagem de ofuscamento de código para prevenir que a engenharia reversa viole a propriedade intelectual dos códigos de aplicações (Collberg *et al.*, 1997; Wang, 2000; Ogiso *et al.*, 2003; Sakabe e Myaji, 2003). Nota-se que esta abordagem assume que é possível para um atacante analisar o agente.

Esta abordagem demanda tempo de execução e largura de banda de comunicação e requer

algumas restrições críticas de tempo [3], mas garante aos agentes a possibilidade de fazer algumas tarefas sensíveis quanto a segurança, sem o perigo de uma exploração imediata do dado sensível por parte do sítio ou plataforma de execução. O difícil é medir por quanto tempo este dado permanecerá protegido.

Collberg *et al.* (1997) afirmam que o ofuscamento de código automático é a técnica mais viável que previne a engenharia reversa maliciosa e com isso garante a confidencialidade dos códigos móveis quando estes estão sendo executados em plataformas não confiáveis. Neste artigo, os autores descrevem uma grande quantidade de transformações de ofuscamento, as classificam como transformações de *layout* ou lexical, transformações de dados, transformações de controle e transformações preventivas. No referido trabalho, as transformações são avaliadas de acordo com a sua potência (quanto uma transformação consegue confundir um leitor humano), sua resistência (quanto uma transformação resistirá a um ataque de uma ferramenta de desofuscamento) e seu custo (quanto tempo/espaço será acrescido a um código ofuscado em relação ao código original). Uma contribuição chave do trabalho de Collberg *et al.* (1997) é o conceito de predicados opacos. Um predicado é opaco se seu resultado é conhecido a priori pelo ofuscador, mas se torna difícil para um desofuscador deduzi-lo. O problema fundamental com os trabalhos de Collberg *et al.* (1997) e Hohl (1998) é que estes trabalhos não suportam uma métrica quantificável. Os critérios apresentados por Collberg et al. estão baseados em métricas de complexidade de software, através de uma avaliação empírica. Wang (2000) aponta que essas métricas não refletem a complexidade real de um programa e que o uso dessas métricas não implica resultados quantificáveis. Segundo Sakabe e Myaji (2003), as técnicas de ofuscamento apresentadas em Collberg *et al.* (1997) não têm uma base teórica consistente e por isso não fica evidente a efetividade das mesmas.

Apresentando resultados teóricos convincentes, Wang *et al.* (2000) propuseram um conjunto de técnicas de ofuscamento que introduz *aliases* não triviais[4] que visam impedir a análise estática[5] de um programa, decompondo, sistematicamente, o fluxo de controle e tornando-o dependente dos dados. Isto é, as análises do fluxo de controle e de dados se tornam co-dependentes, aumentando com isso a complexidade das análises e reduzindo a precisão das análises. Para isto, as transferências de controle de alto nível precisam ser transformadas para serem endereçadas indiretamente através de ponteiros. Como a detecção precisa de *alias* na presença de ponteiros e estruturas recursivas de dados é conhecida como sendo não decidível (Landi, 1992), os autores avaliam a complexidade da sua proposta provando que a análise estática é um problema *NP-hard*. Entretanto, esta abordagem está limitada a análises dentro de procedimentos. Como um programa consiste de muitos procedimentos, se estes estão ou não ofuscados, será preciso uma análise entre procedimentos para entender exatamente o programa. Ogiso *et al.* (2003) propõem técnicas de ofuscamento

[3] Quanto tempo deve durar o intervalo de proteção para permitir que o agente faça algo de útil ?

[4] *Aliases* ocorrem quando dois ou mais nomes referem-se a uma mesma localização de memória.

[5] Análise estática refere-se às técnicas projetadas para extrair informações de uma imagem estática de um programa de computador.

baseadas no uso de ponteiros de funções e vetores de ponteiros para impedir a análise interprocedural.

As duas últimas técnicas apresentadas não podem ser diretamente aplicadas a linguagens orientadas a objetos, especialmente o Java, já que estas técnicas requerem o uso de ponteiros e de procedimentos *goto*, não suportados por estas linguagens. Sakabe e Myaji (2003) apresentam duas técnicas de ofuscamento de código voltadas especificamente para a linguagem Java que usufrui dos conceitos de polimorfismo e de exceções, conforme suportado pelo Java, visando impedir uma análise estática precisa. A avaliação da complexidade dessas técnicas está baseada em resultados teóricos e empíricos, conforme descrita em Sakabe e Myaji (2003).

Enquanto na criptografia tradicional é possível expressar a força da proteção do algoritmo criptográfico, em termos do poder computacional necessário para resolver o problema matemático, os ataques possíveis na abordagem de ofuscamento de código são numerosos e diferem em natureza. A falta de um modelo formal para estimar a força de proteção é o maior problema dessa abordagem.

Alguns autores afirmam que a completa proteção do código contra engenharia reversa maliciosa (abordagem caixa-preta) é um objetivo inatingível (Barak *et al.*, 2001), já que não há nenhum método que torne um programa ilegível. Contudo, os recentes trabalhos descritos em (Wang, 2000; Ogiso *et al.*, 2003; Sakabe e Myaji, 2003) mostram que algum grau de proteção pode ser garantido para um agente ofuscado, tornando esta técnica prática para proteção da propriedade intelectual de um agente.

6.2.9 Considerações sobre as Técnicas para Proteção dos Agentes Móveis

Uma classificação dos mecanismos de segurança para agentes contra o ambiente computacional pode ser feita partindo do seu propósito principal: prevenção ou detecção (Picco, 1998), a saber:

- Os **Mecanismos de Prevenção** tentam tornar mínima a possibilidade de acesso e/ou de modificação dos agentes e se utilizam das seguintes abordagens: *Hardware* seguro, Geração de Chave do Ambiente (Riordan e Schneier, 1998), Computação com Funções Cifradas (Sander e Tschudin, 1998) e Ofuscamento de Código (Hohl, 1998; Collberg *et al.*, 1997; Sakabe e Myaji, 2003).

- Os **Mecanismos de Detecção** tentam descobrir se e quando um ataque foi realizado, após a execução do agente. Esses mecanismos se utilizam das seguintes abordagens: Encapsulamento de Resultado Parcial (Young e Yung, 1997; Yee, 1997; Karjoth *et al.*, 1998; Karnik, 1998; Corradi *et al.*, 1999a), Contêiner de Dados Somente para Inclusão e Vetor de Dados Direcionados (Karnik, 1998), Registro de Itinerário Mútuo (Roth,

1998), Registro de Itinerário com Replicação e Votação (Schneider, 1997) e Rastros Criptográficos (Vigna, 1998a).

Conforme analisado neste capítulo, é um problema bastante difícil e ainda sem solução adequada a proteção dos agentes móveis contra as ameaças de ambientes computacionais maliciosos já que os agentes são executados nestes ambientes que podem ter acesso aos seus dados e até mesmo modificá-los. A segurança de agentes móveis envolve tanto a integridade do código do agente e do seu estado, quanto a revelação de informações do código (p.ex, violam a propriedade intelectual) e do estado do agente. Através de técnicas de assinatura digitais, é possível proteger a integridade de códigos de agentes e de seus valores de atributos originais, porém, no caso de agentes *multi-hop*, a integridade dos dados gerados nas plataformas visitadas é mais difícil de ser garantida.

As abordagens de prevenção baseadas em *Hardwares* Seguros garantem a integridade do agente e de seu estado, porém estas não são práticas devido aos elevados custos associados à necessidade desses *hardwares*. Já a abordagem de Computação com Funções Cifradas (Sander e Tschudin, 1998), que visa garantir a confidencialidade do agente durante a sua computação, apesar de serem técnicas de prevenção baseadas em software, até o momento não apresentam soluções comprovadamente eficientes e gerais. Portanto, não são consideradas práticas. Devido à dificuldade em medir a eficiência e o tempo de proteção da abordagem de ofuscamento de código, a aplicabilidade desta abordagem se mostra limitada a aplicações que exigem uma segurança branda (*soft security*).

As técnicas de detecção são as que se mostram mais adequadas de serem implantadas. Porém, as limitações apontadas nas técnicas de Registro de Itinerário Mútuo, Registro de Itinerário com Replicação e Votação e Rastro de Execução precisam ainda ser cuidadosamente analisadas para que estas técnicas sejam consideradas plenamente aplicáveis. Além disso, estas técnicas não tratam do problema da confidencialidade durante a computação de um agente, pois seus objetivos estão ligados à detecção de plataformas maliciosas quando estas afetam a integridade de agentes móveis. Os problemas e limitações apresentados nas técnicas acima são ainda maiores quando se leva em conta aplicações em sistemas abertos que anseiam por escalabilidade, portabilidade e interoperabilidade, já que estas aplicações envolvem vários domínios administrativos e tecnologias heterogêneas.

As abordagens de Encapsulamento de Resultados Parciais, que dependem da plataforma para executar o encapusulamento e as abordagens dos Contêineres de Dados Somente Leitura e de Dados Direcionados, apesar de suas limitações, são as que apresentam melhores resultados.

6.3 $MASS_{ma}$: Esquema de Segurança para Proteção de Agentes Móveis

Conforme introduzido na seção 3.8, o esquema de segurança proposto para proteção dos agentes móveis — $MASS_{ma}$— foi projetado para um modelo de agentes que permite múltiplos-saltos e itinerários livres. Além disso, o $MASS_{ma}$ faz uso da infra-estrutura de chave pública do SPKI (Elisson, 1999) e do conceito de federações SPKI (Santin, 2004) que fornecem meios para garantir a escalabilidade e a flexibilidade necessárias às aplicações distribuídas.

A Figura 6.5 apresenta os procedimentos suportados pelo esquema de segurança proposto, composto de técnicas de prevenção e de detecção. Antes de enviar um agente móvel, a plataforma-fonte pode verificar, com o auxílio de informações públicas, a reputação da plataforma-destino (objeto 1, Figura 6.5). Esta reputação é definida no *MASS* por um mecanismo de controle social que usufrui da infra-estrutura das federações de serviço SPKI. Apesar da autenticação de agentes móveis ser uma técnica que primeiramente visa proteger as plataformas contra agentes maliciosos, esta pode ser usada para detectar modificações não autorizadas em um agente no decorrer de suas viagens e para auxiliar o processo de identificação de plataformas maliciosas visitadas (objeto 2, Figura 6.5). Quando o interesse é a proteção do estado de um agente, o programador pode selecionar uma ou mais técnicas, baseadas em repositórios de dados seguros, para criação de um agente móvel protegido (objeto 3, Figura 6.5). Durante o processo de autenticação do agente, quando uma violação da integridade do agente for detectada, o mecanismo que aplica o controle social das plataformas deve ser acionado para que a identificação de plataformas maliciosas seja possível (objeto 4, Figura 6.5).

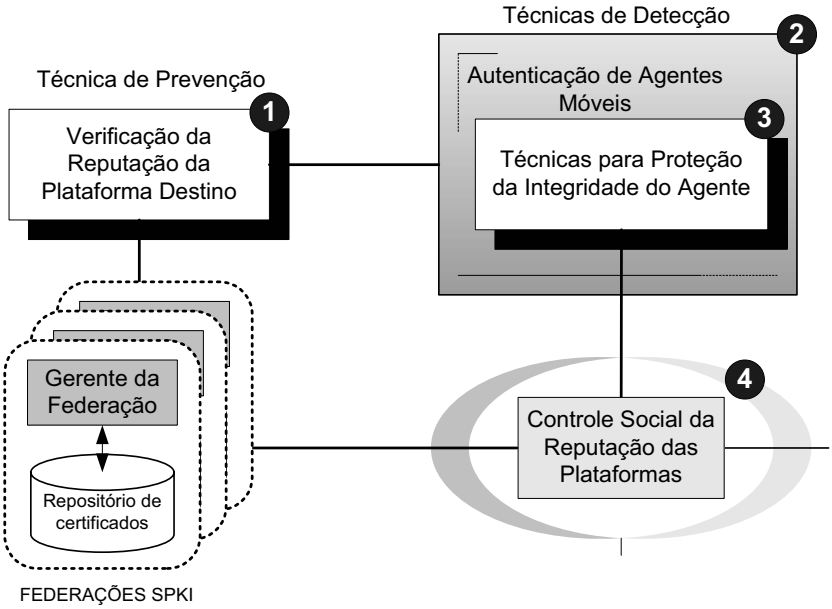


Figura 6.5: $MASS_{ma}$: Esquema de Segurança para Proteção de Agentes Móveis

Visando detectar possíveis alterações, remoções ou inserções de dados em um agente mó-

vel (violação da integridade), três técnicas estão disponíveis no $MASS_{ma}$: uma para proteção do código do agente **Assinatura do Código**, outra para proteção dos dados imutáveis do agente — **Repositório de Dados Somente-Leitura** — e a terceira para o resultados parciais recolhidos durante as viagens do agente — **Repositório Seguro de Resultados Parciais**. Um quarta técnica, que visa não só a integridade de dados, mas também a confidencialidade de alguns dados para que não sejam revelados a plataformas não autorizadas, também compõe o esquema proposto — **Repositório de Dados Direcionados**. O esquema proposto irá auxiliar o programador do agente na construção de um agente móvel protegido e ainda auxiliar as plataformas visitadas na verificação da integridade do agente (código e repositórios), através de um **Autenticador *Multi-Hop***.

6.3.1 Suportes e Premissas

Qualidade de Proteção - *QoP*

No esquema proposto, uma plataforma, ao criar um agente móvel em seu contexto, antes de dispará-lo para a primeira plataforma a ser visitada, deve definir a **qualidade de proteção** requerida para o agente, chamada de *QoP*, é um atributo somente-leitura único que expressa os mecanismos de segurança que devem ser usados pelas plataformas visitadas pelo agente. A Tabela 6.3.1 apresenta os possíveis valores que podem compor o atributo *QoP* e os objetivos de segurança associados ao mecanismo de segurança correspondente. O atributo *QoP* é composto por um ou mais valores (mecanismos) descritos na tabela. Os mecanismos serão descritos nas próximas seções.

Valores de *QoP*	Objetivo de Segurança
Verifica_Reputacao	Prevenir que um agente móvel seja enviado para uma plataforma não confiável
Codigo_Assinado	Proteger a integridade do código de um agente
RepositorioSL	Proteger a integridade dos dados somente-leitura de um agente
RepositorioDD	Prevenir a revelação não autorizada de dados direcionados a plataformas pré-definidas de um agente móvel
RepositorioRP	Proteger a integridade dos resultados parciais coletados e carregados por um agente móvel
Registrador_Caminhos	Registrar o caminho percorrido por um agente móvel e proteger a integridade do objeto registrador de caminhos

Tabela 6.1: Possíveis Valores para Compor o Atributo *QoP* de um Agente Móvel

Premissas do Esquema Proposto

Além das suposições e premissas gerais introduzidas na seção 3.8.2, o $MASS_{ma}$ é construído tomando como base as seguintes premissas:

- a plataforma de origem centraliza a coordenação do protocolo de identificação de plataformas maliciosas quando uma violação do seu agente for detectada;

- se uma plataforma não cumprir com os procedimentos necessários para atender a qualidade de proteção (*QoP*) requerida pelo agente, esta é considerada maliciosa;
- não pode haver conluio entre plataformas visando violações de segurança;
- quando uma plataforma detectar uma violação da integridade do agente, seja do código ou do estado do agente, esta deve imediatamente suspender o agente e informar à plataforma de origem da violação sofrida, caso contrário, a plataforma é considerada maliciosa.

Notação Usada no Esquema Proposto

A Tabela 6.2 resume as notações usadas no esquema proposto. Assume-se no esquema que, dada a assinatura $SIG_i(m)$, concretizada através de funções criptográficas, qualquer plataforma poderá extrair m. Ou seja, $SIG_i(m)$ é a união da assinatura com o dado cifrado. Além disso, a identidade do assinante P_i pode ser deduzida examinando a assinatura $SIG_i(m)$.

P_o	Identidade da plataforma de origem
$P_i, 1 \leq i \leq n$	Identidades das plataformas visitadas por um agente móvel
r_i	Número aleatório gerado pela plataforma P_i
$ENC_o(key_i)$	Chave temporária (key_i) cifrada com a chave pública de P_o
$ENC_{key_i}(m)$	Mensagem m cifrada com chave secreta temporária criada em P_i
$SIG_i(m)$	Assinatura da plataforma P_i sobre a mensagem m
$H(m)$	Uma função *hash one-way* [6] (p.ex, SHA-1)
$P_1 -> P_2 : m$	Plataforma P_1 enviando a mensagem m para P_2
$creds$	Credenciais de um agente móvel

Tabela 6.2: Notação Criptográfica

6.3.2 Estrutura Proposta para um Agente Móvel

Conceitualmente, um agente sendo um objeto Java, consiste de código (representado por um conjunto de métodos em sua classe Java) e estado, isto é, atributos da sua classe. Para implantação das técnicas que compõem o $MASS_{ma}$, partes do estado do agente precisam ser distintos, conforme ilustrado na Figura 6.6.

A maioria dos sistemas de agentes móveis usa apenas a assinatura do código do agente pelo proprietário para expressar a posse de um agente móvel (em nome de quem este agente atua). Porém, segundo Roth (2001), isto não é suficiente para distinguir uma instância de um agente de outra instância. Segundo o autor, o proprietário do agente deve assinar um dado estático do agente que pode incluir o código do agente e uma redundância suficiente para distinguir entre duas instâncias de um mesmo agente (classe). No esquema $MASS_{ma}$, após a criação do programa do agente, o proprietário deve gerar o dado estático do agente, chamado de credenciais do agente (**objeto *Credenciais***) para distinguir instâncias de um mesmo agente. O *hash* do objeto *Credenciais* ($H(creds)$) serve como um identificador único do

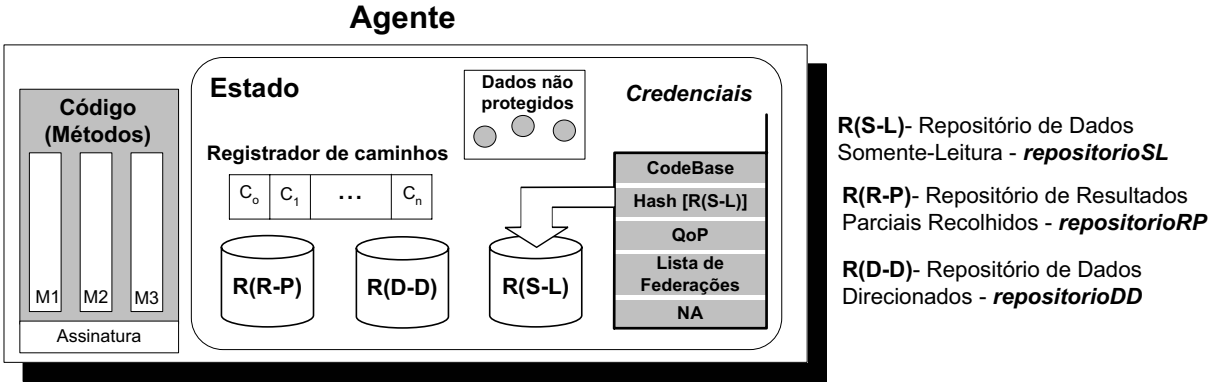


Figura 6.6: Estrutura Proposta para um Agente Móvel

agente que deve ser agregado aos repositórios de dados do agente para garantir que a instância do agente seja única. Cada agente deve carregar suas credenciais assinadas como parte do seu estado dentro de um repositório de dados somente-leitura, chamado de *repositorioSL*. Os objetos *Credenciais* são definidos por cinco campos (ver Figura 6.6).

***Codebase* do agente.** Especifica a localização do servidor das classes que um agente pode requerer. Se a plataforma corrente do agente necessita obter o código de alguma classe necessária ao agente, esta deve contactar a entidade indicada no *codebase* e usar um protocolo autenticado e seguro para carregar essas classes. Vale ressaltar a importância de se proteger o *codebase* para que este não aponte para uma entidade não confiável.

***Hash* dos dados do repositorioSL.** O principal objetivo deste resumo criptográfico é o de ligar o objeto *Credenciais* com os dados iniciais do agente, armazenados no repositório de dados somente-leitura (*repositorioSL*). Propõe-se que o valor do *hash* criptográfico dos dados somente-leitura[7] seja também armazenado no objeto *Credenciais*. Esta ligação é útil para detectar ataques em que uma plataforma maliciosa reutiliza as credenciais de um agente visitante em um outro agente.

Qualidade de Proteção (*QoP*). Atributo que identifica a qualidade de proteção requerida que deve ser atendida por todas as plataformas visitadas pelo agente.

Lista de Federações SPKI. Atributo que indica as federações SPKI cujas plataformas membros estão autorizadas a executar o agente.

Número Aleatório. Por fim, um número aleatório (*NA*), grande o suficiente para não ser reproduzido duas vezes pelo proprietário do agente, deve ser incluído no objeto *Credenciais*.

Na estrutura proposta para um agente móvel, ilustrada na Figura 6.6, um objeto registrador de caminhos e três objetos opcionais, chamados de repositórios de dados (R(S-L),

[7] Somente os dados estáticos referentes a aplicação, sem o objeto *Credenciais*.

R(R-P)e R(D-D)), podem fazer parte ainda do estado do agente. O programador do agente deve criar o registrador de caminhos se este deseja que um histórico das plataformas visitadas seja carregado pelo agente (mais detalhes na seção 6.4). Já os repositórios de dados são utilizados para armazenar de forma segura os dados transportados/coletados pelo agente.

Nas seções seguintes, serão discutidas em detalhes as técnicas de prevenção e detecção, apresentadas na Figura 6.5, que compõem o esquema de segurança para proteção de agentes móveis.

6.3.3 Verificação da Reputação da Plataforma Destino

De forma a prevenir o envio de um agente móvel para uma plataforma que não é membro de uma das federações autorizadas a receber o agente ou que não seja considerada confiável, o proprietário do agente pode, através do atributo *QoP*, expressar que a verificação da reputação das plataformas que irão receber o seu agente deve ser realizada. Com esta verificação, é possível auxiliar o agente no estabelecimento da confiança nas plataformas a serem visitadas. É responsabilidade das plataformas que irão enviar o agente, cumprir com a qualidade de proteção requerida e definir o nível de confiança do agente na próxima plataforma a ser visitada.

O mecanismo proposto de verificação da reputação depende de políticas de controle social que, diante de detecções de violações do agente, procura identificar plataformas suspeitas e atualizar a reputação das plataformas divulgadas em listas armazenadas nas federações. Duas listas foram definidas para o controle social no $MASS_{ma}$: uma *lista negra*, identificando plataformas maliciosas que fizeram ou fazem parte da federação; e uma *lista vermelha*, identificando a reputação de cada plataforma ligada à federação (plataformas com administradores/principais registrados nas federações). As federações devem fornecer mecanismos para a consulta automática destas listas de reputação. O mecanismo de controle social proposto é descrito na seção 6.5. Na verificação da reputação, somente a *lista negra* será analisada.

Dois passos foram definidos para a verificação (ver Figura 6.7): o primeiro solicita as *listas negras* armazenadas nas federações e o segundo analisa estas listas e define se a próxima plataforma a ser visitada é ou não confiável. Uma plataforma é considerada *não confiável* quando esta se encontra em uma das listas negras obtidas. Caso uma plataforma destino seja considerada *não confiável*, o agente deve ser enviado para a sua plataforma de origem. O passo 1 da Figura 6.7 pode adicionar custo no desempenho de algumas aplicações. Neste sentido, definiu-se que as listas enviadas pelas federações podem ser armazenadas por um dado período de tempo nas plataformas, eliminando os custos de comunicação a cada envio de agente na plataforma. Após a expiração deste tempo, estas listas precisam ser atualizadas.

O mecanismo proposto para verificação da reputação da plataforma destino quando combinado ao protocolo de autenticação mútua de plataformas, descrito no Capítulo 4, torna ainda mais eficiente o estabelecimento da confiança de um agente em uma plataforma.

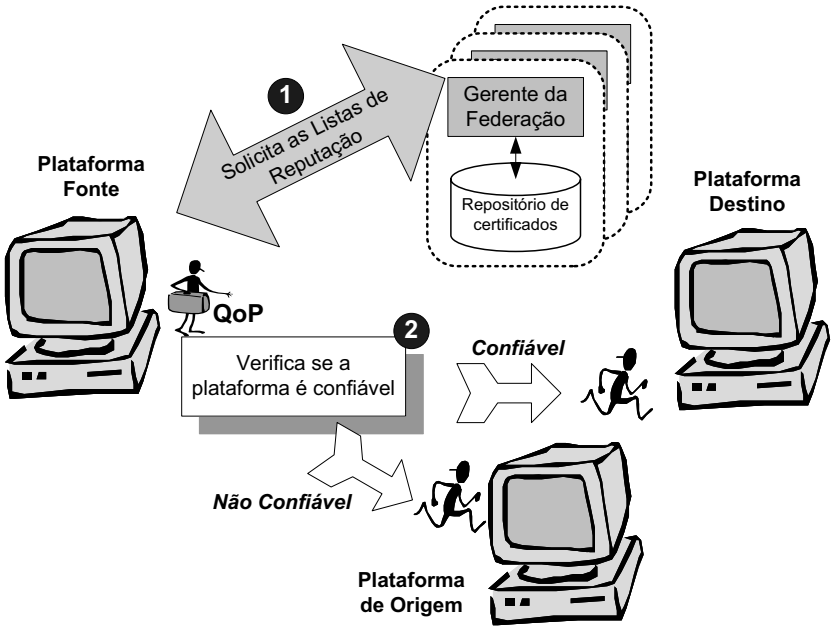


Figura 6.7: Procedimentos para Verificação da Reputação da Plataforma Destino

6.3.4 Assinatura do código e o Repositório Somente Leitura

Na maioria das plataformas de agentes móveis baseadas na linguagem Java, quando um agente migra, não se transfere o código junto com o estado. Em muitos casos, é possível que a plataforma destino já tenha disponível localmente alguma das classes que o agente usará. Com isso, o código é apenas solicitado sob demanda a partir do *codebase* do agente, se o mesmo não pode ser suprido pela máquina virtual local. Entretanto, isto depende de como o sistema de agentes implementa a transferência de agentes, visto que em algumas aplicações em que não se deseja ter interações com a plataforma de origem do agente ou com outra que este confia, a transferência do código se faz necessária junto com o estado do mesmo. No esquema proposto, um proprietário deve assinar digitalmente o código do seu agente para proteger a sua integridade — $SIG_o(codigo)$. Sempre que este código for solicitado sob demanda ou for serializado para ser transportado pela rede, esta assinatura deverá acompanhá-lo. Desta forma, modificações feitas no código do agente podem ser facilmente detectadas por qualquer plataforma visitada pelo agente. Quando uma plataforma detectar uma modificação, esta deve avisar à plataforma de origem para que providências sejam tomadas (controle dinâmico da reputação das plataformas).

Além do código, no $MASS_{ma}$, o proprietário do agente deve ainda assinar todos os dados originais indicados pelo programador do agente como sendo somente-leitura e armazenados no repositório de nome ***repositorioSL*** — $SIG_o(dados\ do\ repositorioSL)$. A técnica proposta está baseada no Contêiner Somente-Leitura, proposto por Karnik e Tripathi (2001). Além dos dados específicos de cada aplicação, o repositório somente leitura conterá o objeto **Credenciais** do agente. Além disso, o *hash* dos dados do *repositorioSL* é incluído no objeto que contém as credenciais do agente quando este é assinado pelo dono do agente. Desta forma,

uma ligação a prova de modificações é criada entre o estado somente-leitura e as credenciais do agente. Qualquer plataforma visitada pelo agente pode verificar a integridade deste repositório e a sua associação ao agente recebido. Para isto, basta a plataforma visitada obter a chave pública da plataforma de origem (proprietário do agente), decifrar a assinatura, recalcular o *hash* dos dados armazenados no *repositorioSL* e comparar os resultados. Qualquer violação detectada deve ser repassada para a plataforma de origem.

6.3.5 Repositório Seguro de Resultados Parciais

No esquema proposto, os dados sensíveis, gerados em uma plataforma, devem ser armazenados em um repositório seguro de resultados parciais, chamado de ***repositorioRP***, que será carregado pelo agente, para que possíveis modificações de plataformas maliciosas possam ser detectadas.

No $MASS_{ma}$, os protocolos criptográficos usados pelas plataformas de agentes móveis para inserir e proteger um resultado no *repositorioRP* são semelhantes aos protocolos P1 e P2 definidos por Karjoth *et al.* (1998) e descritos na seção 6.2.1 (Famíla KAG de protocolos). Algumas adaptações e modificações foram realizadas nos protocolos P1 e P2 da Família KAG e estas serão discutidas a seguir. Seguindo a notação apresentada na Tabela 6.3, as Figuras 6.8, 6.9 e 6.10 descrevem os procedimentos necessários para inicialização do *repositorioRP*, para inserção de um resultado parcial e para verificação da integridade do *repositorioRP*, respectivamente.

$rp_i, 1 \leq i \leq n$	Resultado parcial obtido na plataforma P_i
$RP_i, 1 \leq i \leq n$	Resultado parcial protegido obtido na plataforma P_i
$h_i, 1 \leq i \leq n$	Valor associado com RP_i para verificação da integridade (relação de encadeamento)
$H(creds)$	Hash do objeto Credenciais (identificador único do agente)

Tabela 6.3: Notação do Esquema Repositório Seguro de Resultados Parciais

6.3.5.1 Inicialização do Repositório de Resultados Parciais

Para iniciar o *repositorioRP*, o proprietário do agente (plataforma de origem P_o) gera um número aleatório r_o e computa o valor de h_o, aplicando uma função *hash one-way* sobre r_o e a identidade da primeira plataforma a ser visitada (P_1) (passo 1 da Figura 6.8). Este h_o é o valor âncora da relação de encadeamento dos resultados parciais recolhidos. O proprietário deve ainda cifrar o valor r_o com a sua própria chave pública. Em seguida, o resultado deste ciframento mais o identificador único do agente e a relação de encadeamento h_o são assinados pelo proprietário do agente, criando o resultado parcial protegido RP_o (passo 2, Figura 6.8). No esquema proposto, o identificador único do agente é o valor resultante da aplicação

da função *hash* sobre o objeto *Credenciais*; já nos protocolos da família KAG, este identificador é um *token* emitido pela plataforma de origem. Finalmente, no passo 3, o resultado parcial é inserido no *repositorioRP* (RP_o) e enviado para a plataforma P_1.

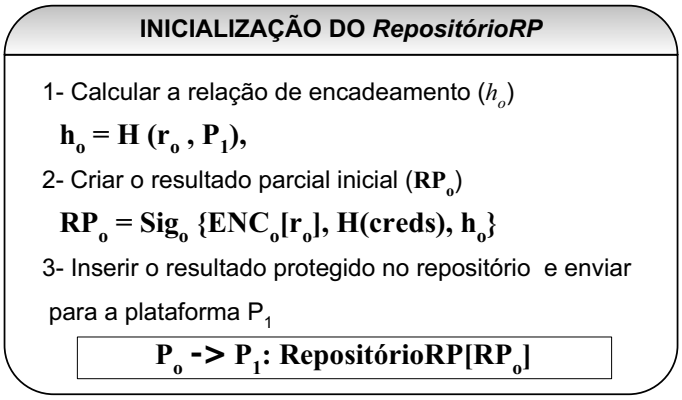


Figura 6.8: Inicialização do Repositório de Resultados Parciais

Ao contrário do que ocorre nos protocolos da Família KAG, em todos os protocolos de inserção de resultados descritos, a seguir, o processo de inicialização do *repositorioRP* é o mesmo (Figura 6.8). No protocolo P2 da Família KAG, o resultado parcial inicial é calculado de forma diferente do definido no protocolo P1 — $RP_o = ENC_o(SIG_o(rp_o), r_o), h_o$. Conforme constatado em Roth (2001), o procedimento de inicialização do protocolo P2 da família KAG apresenta um problema não tratado. Uma plataforma não tem como saber qual é a plataforma de origem do agente uma vez que a assinatura do resultado inicial está escondida pelo ciframento. Este problema é contornado, no presente trabalho, pelo protocolo de iniciação apresentado na Figura 6.8.

6.3.5.2 Inserção de Resultados Parciais

A Figura 6.9 sintetiza os três protocolos para inserção de resultados parciais no ***repositorioRP***, suportados pelo esquema $MASS_{ma}$. O proprietário do agente deve definir qual protocolo será usado pelas plataformas de agentes para inserir os resultados de acordo com a necessidade de cada aplicação e informar o protocolo usado às plataformas a serem visitadas.

Protocolo A

O primeiro protocolo proposto para inclusão de resultados parciais está baseado no protocolo P1 da família KAG (Karjoth *et al.*, 1998) — *Protocolo de Assinatura Digital Encadeada Verificável Publicamente*. O primeiro passo do protocolo descrito na Figura 6.9.a e o passo do protocolo P1 são idênticos — calcular o valor *hash* sobre o estado parcial anterior mais a identidade da próxima plataforma a ser visitada. O uso de h_i, no passo 1, tem dois objetivos. Primeiro, este liga o resultado obtido anteriormente (RP_{i-1}) com o resultado parcial corrente (RP_i). Logo, RP_{i-1} não pode ser modificado sem afetar RP_i. A plataforma P_i, em uma segunda visita do agente, também não pode modificar seu próprio resultado sem invalidar a

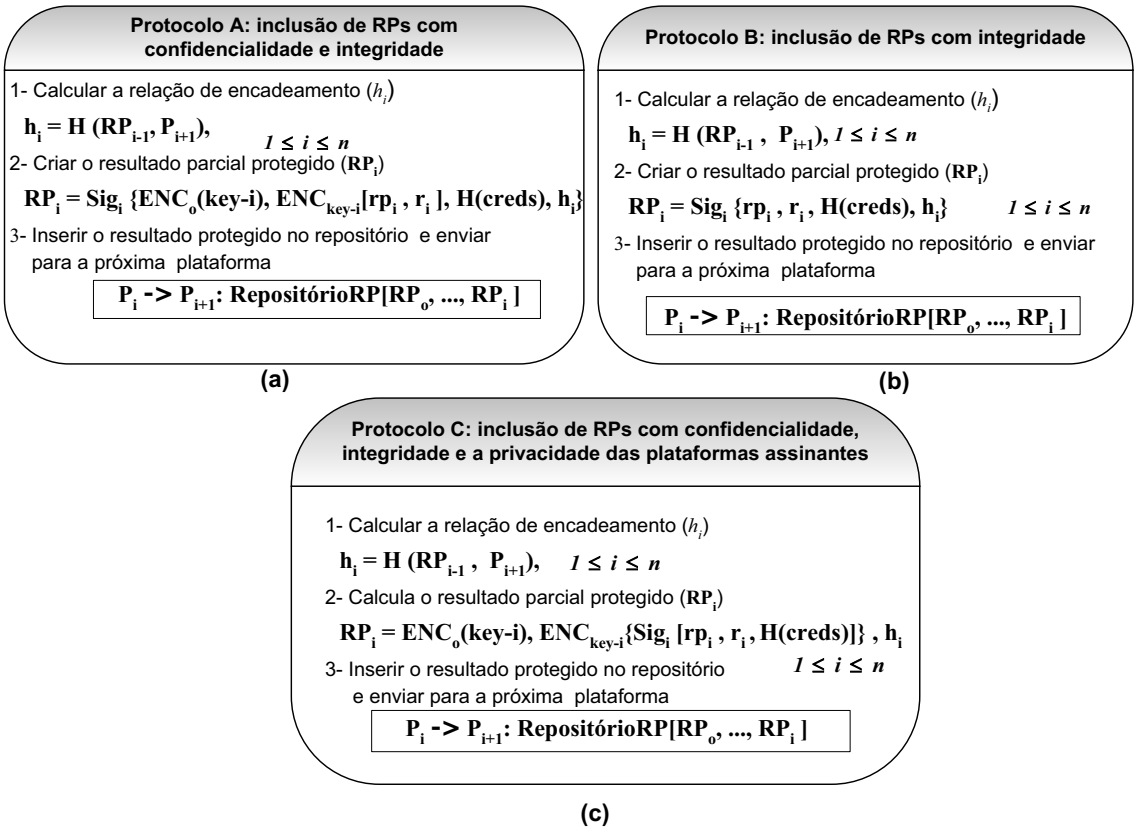


Figura 6.9: Protocolos para Inserção de Resultados Parciais

cadeia em RP_i. Segundo objetivo, a inclusão da identidade da próxima plataforma no cálculo de h_i garante ainda que somente P_{i+1} pode adicionar o próximo resultado parcial.

No passo 2 do Protocolo A, visando manter a confidencialidade de alguns dados sensíveis gerados nas plataformas e que serão carregados pelos agentes, cada plataforma deve cifrar, com a chave secreta temporária, o resultado parcial concatenado a um número aleatório gerado pela própria plataforma. Esta chave secreta temporária, criada em cada plataforma P_i, é cifrada com a chave pública da plataforma de origem para que somente esta plataforma tenha acesso à chave temporária e, conseqüentemente, aos dados. Para construir o resultado parcial protegido (RP_i), a plataforma deve assinar os resultados dos ciframentos, concatenado com a relação de encadeamento (h_i) e com o identificador único do agente ($H(creds)$).

Este passo é que difere o protocolo proposto do protocolo P1 da família KAG. A modificação está na inclusão do identificador do agente ($H(creds)$) — uma redundância criptográfica, que passa a ser anexada ao resultado parcial encapsulado em cada plataforma. Esta modificação tem por objetivo relacionar cada resultado parcial coletado com a instância do agente que está em execução, contornando assim o ataque descrito na seção 6.2.1 ao qual os protocolos P1 e P2 da família KAG são vulneráveis. Além disso, a cifragem de chave secreta dos dados e a distribuição da chave temporária, usando chave pública (esquema híbrido de cifragem), reduz os custos computacionais quando comparados com os custos dos protocolos da família KAG, que utiliza cifragem de chave pública.

Objetivos de Segurança	Protocol A	Protocolo B	Protocolo C
Confidencialidade dos resultados inseridos	sim	não	sim
Integridade em avanço (forte)	sim	sim	sim
Não-repudiação da inserção de resultados	sim	sim	sim
Integridade verificável publicamente	sim	sim	não
Privacidade do assinante do resultado	não	não	sim
Resistência a inserção de resultados falsos	sim	sim	sim
Resistência a remoção total dos dados	não	não	não

Tabela 6.4: Comparação dos Objetivos de Segurança dos Protocolos Propostos

Protocolo B

No protocolo A, os dados cifrados podem impor um custo ao desempenho das aplicações. Examinando este problema, propõe-se um segundo protocolo (Protocolo B) para inserção de resultados parciais (Figura 6.9.b), que não exige o ciframento dos dados coletados. Este protocolo é semelhante ao anterior e só difere no passo 2, em que os dados não são cifrados.

Protocolo C

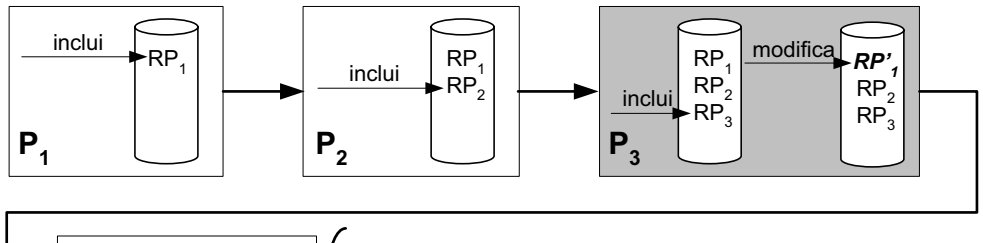
O terceiro protocolo proposto está baseado no Protocolo P2 da família KAG — *Protocolo de Assinatura Digital Encadeada com Privacidade do Assinante*. O protocolo P2 é uma variação do protocolo P1, em que a ordem da cifragem e da assinatura digital são trocadas visando esconder a identidade das plataformas que inseriram resultados, mantendo ainda o objetivo pela integridade. Entretanto, a integridade dos resultados inseridos não poderá mais ser verificada publicamente. No protocolo C, ilustrado na Figura 6.9.c, o passo 1 é idêntico ao protocolo P2 da Família KAG, já o passo 2 inclui a redundância necessária para associar cada resultado encapsulado com a instância do agente e o uso do esquema híbrido de cifragem. Porém, conforme discutido anteriormente, o processo de inicialização do Protocolo C difere do processo do protocolo P2 da família KAG.

A Tabela 6.4 compara os objetivos de segurança de cada protocolo proposto, tendo como base os objetivos apresentados em Karjoth *et al.* (1998). Conforme indicado na referida Tabela, a remoção total de resultados parciais coletados não é detectada em nenhum dos protocolos. Segundo Karjoth *et al.* (1998); Karnik e Tripathi (2001); Roth (2001), a remoção total de resultados é o problema mais difícil a ser tratado na preservação do estado da cadeia de resultados parciais encapsulados. Quando duas plataformas maliciosas conspiram[8], estas podem remover todos os dados inseridos nos sítios visitados entre as duas plataformas maliciosas. O mesmo acontece quando um agente visita uma plataforma maliciosa mais de uma vez.

[8]Duas plataformas conspiram quando estas trocam segredos usados na construção da relação de encadeamento.

6.3.5.3 Verificação da Integridade do *repositorioRP*

Conforme pode ser observado na Tabela 6.4, nos protocolos A e B, a integridade em avanço dos resultados parciais encapsulados pode ser verificada por qualquer plataforma visitada pelo agente. Cada plataforma examinadora deve verificar: (1) se a assinatura de cada RP_i é valida; (2) se cada RP_i está ligado com o identificador único do agente; e (3) se a relação de encadeamento é assegurada em cada ligação. A Figura 6.10 ilustra um exemplo, em que a plataforma P_4 verifica a integridade do *repositorioRP* carregado por um agente que percorreu as plataformas P_1, P_2 e P_3, respectivamente. No exemplo ilustrado, a plataforma P_3, após inserir o seu resultado parcial, substitui o resultado RP_1, inserido pela plataforma RP_1, por RP'_1. O procedimento necessário para verificar a integridade dos resultados parciais protegidos (*RPs*) está também descrito na Figura 6.10. No passo 1, o primeiro resultado parcial, cuja integridade será verificada, é obtido do *repositorioRP* (RP_3). No passo 2, usando a chave pública da plataforma P_3, (X_3, Y_3, h_3) é obtido, onde X_3 e Y_3 são strings não interpretáveis (resultantes do processo de cifragem — ver Figura 6.9.a) e h_3 é a relação de encadeamento em P_3. Nos passos 3 e 4, a relação de encadeamento é re-calculada e comparada com o valor obtido no passo 2. Então confirma-se que a relação de encadeamento é válida em RP_3. No passo 5, inicia-se a verificação de RP_2. Como h_2 foi obtido usando RP_1, quando $H(RP'_1, P4)$ for calculado (passo 7), será assim comprovado que o repositório foi violado em RP_1 e, devido à relação de encadeamento entre os resultados, todos os resultados inseridos após RP_1 também estão comprometidos. Com esta verificação, é possível detectar a violação, porém não se tem o conhecimento preciso de qual foi a(s) plataforma(s) maliciosa(s).



Plataforma P_4 verificando o repositorioRP

- $RP_3 = Sig_3\{X_3, Y_3, H(creds), h_3\}$ (1)
- $K_{3\text{-pub}}(RP_3) \Rightarrow X_3, Y_3, H(creds), h_3$ (verifica a ligação com o agente) (2)
- Calcula $h_3 = H(RP_2, P_4)$ (3)
- Compara o valor de h_3 obtido no passo 2 com o calculado no passo 3. RP_3 está ok! (4)
- $RP_2 = Sig_2\{X_2, Y_2, H(creds), h_2\}$ (5)
- $K_{2\text{-pub}}(RP_2) \Rightarrow X_2, Y_2, H(creds), h_2$ (6)
- Calcula $h_2 = H(RP'_1, P_3)$ (7)
- Compara o valor de h_2 obtido no passo 2 com o calculado no passo 3. Como a relação de encadeamento (h_2) foi calculada com base em RP_1 e não com RP'_1, então:

$$H(RP'_1, P_3) \neq H(RP_1, P_3)$$ --- repositório foi violado! (8)

Figura 6.10: Exemplo de Verificação da Integridade do *repositorioRP*

6.3.6 Repositório Seguro de Dados Direcionados

O esquema de segurança, proposto neste trabalho, suporta ainda um repositório de dados direcionados, chamado de ***repositorioDD***, baseado na técnica implementada na plataforma Ajanta que possibilita uma revelação seletiva do estado de um agente, na qual o programador do agente pode implementar um vetor de dados direcionados em que cada entrada do repositório tenha plataformas específicas como destinatárias (Karnik, 1998). Esta técnica exige que as plataformas definidas como receptoras de informações no repositório estejam pré-determinadas. Conforme discutido na seção 6.2.2, a técnica de vetor de dados direcionados, proposta por Karnik (1998), possui algumas vulnerabilidades que permitem que dados sejam revelados a plataformas maliciosas (Roth, 2001). Visando contornar estas vulnerabilidades, propõe-se neste trabalho um aprimoramento desta técnica com a inclusão de uma redundância criptográfica que liga cada entrada do *repositorioDD* com a instância do agente móvel correspondente.

A Figura 6.11 ilustra um exemplo, onde o programador do agente cria o *repositorioDD* com duas entradas de dados direcionados (DD_1 e DD_3), para as plataformas P_1 e P_3, respectivamente. Para garantir a confidencialidade dos dados direcionados, o programador, com a chave pública da plataforma receptora, deve cifrar a chave secreta temporária criada pela plataforma de origem e então cifrar, com a chave temporária, o dado concatenado com o identificador único da instância do agente. A plataforma de origem deve ainda assinar este valor para garantir a sua integridade (ver Figura 6.11). Quando o agente é recebido em uma plataforma receptora (plataforma P_1 da Figura 6.11), antes de continuar a execução do agente, a plataforma deve verificar a origem deste dado, sua integridade e se este está realmente associado ao agente recebido. Para isto, os passos ilustrados na Figura 6.11 devem ser seguidos.

6.4 Autenticação de Agentes Móveis

Visando prover segurança para as plataformas de agentes móveis, em Wangham *et al.* (2003), foi proposto um autenticador de agentes *multi-hop* que visa estabelecer a confiança em um agente móvel. Este mecanismo auxilia não só a proteção da plataforma mas também a proteção dos agentes móveis, pois detecta as possíveis violações da integridade do código e do estado do agente (repositórios de dados e registrador de caminhos) e ajuda ainda no processo de identificação de plataformas maliciosas. A figura 6.12 ilustra os passos do autenticador *multi-hop* que auxiliam tanto a verificação da integridade do agente quanto o estabelecimento da confiança neste agente (proteção da plataforma). Todos os passos do autenticador proposto são opcionais. A definição de quais passos serão adotados para a autenticação de um dado agente dependerá da qualidade de proteção exigida pelo proprietário do agente e expressa no atributo *QoP*. Todas as plataformas visitadas deverão atender aos

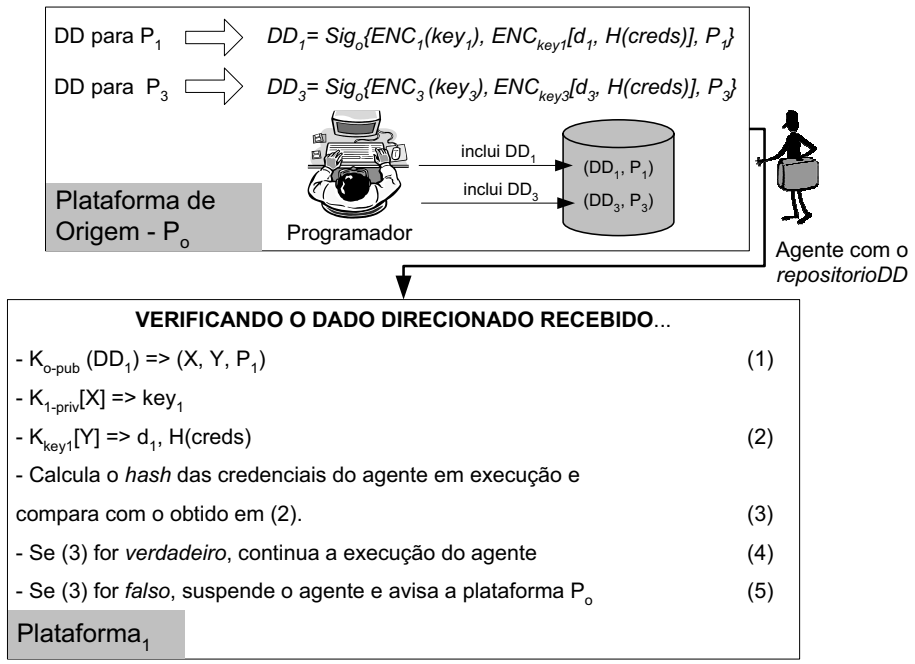


Figura 6.11: Inclusão e Verificação de Dados Direcionados no *repositorioDD*

requisitos mínimos de segurança apontados no atributo *QoP*, caso contrário, serão consideradas maliciosas.

Nos **passos 1 e 2** da Figura 6.12, uma plataforma, ao receber um agente móvel, deve, através da verificação da assinatura do código e do *repositorioSL*, confirmar que este agente não foi corrompido e confirmar a sua associação a um principal, seu proprietário. Desta forma, modificações feitas por plataformas maliciosas no código e/ou nos dados somente-leitura (*repositórioSL*) podem ser facilmente detectadas por qualquer plataforma visitada pelo agente. Quando assim for especificado no atributo *QoP*, a integridade dos dados do *repositorioDD* e do *repositorioRP* que estão sendo carregados pelo agente deve ser verificada (**passo 3**). Quando uma plataforma detectar uma modificação, esta deve avisar à plataforma de origem para que as providências necessárias do mecanismo de controle social de reputação sejam tomadas (**passo 6**).

Visando conferir o percurso de viagens do agente, a plataforma destino pode analisar o histórico do caminho do agente e estabelecer o nível de confiança no agente (**passo 5**). Conforme descrito no capítulo 4, para criar o histórico de caminhos de um agente, cada plataforma visitada deve adicionar em um objeto, *registrador de caminhos*, uma entrada contendo sua identidade (representada por sua chave pública) e a identidade da próxima plataforma a ser visitada, formando a história do caminho percorrido pelo agente (ver Figura 6.13). Como o objeto *registrador de caminhos* está contido no estado do agente (será transportado junto com agente), os dados armazenados neste objeto são considerados sensíveis, então, sua integridade deve ser preservada. De forma a proteger o *registrador de caminhos*, as entradas devem ser inseridas conforme descrito pelo **Protocolo B** do repositório de resultados parciais e ilustrado na Figura 6.13. Desta forma, qualquer plataforma pode verificar a integridade das

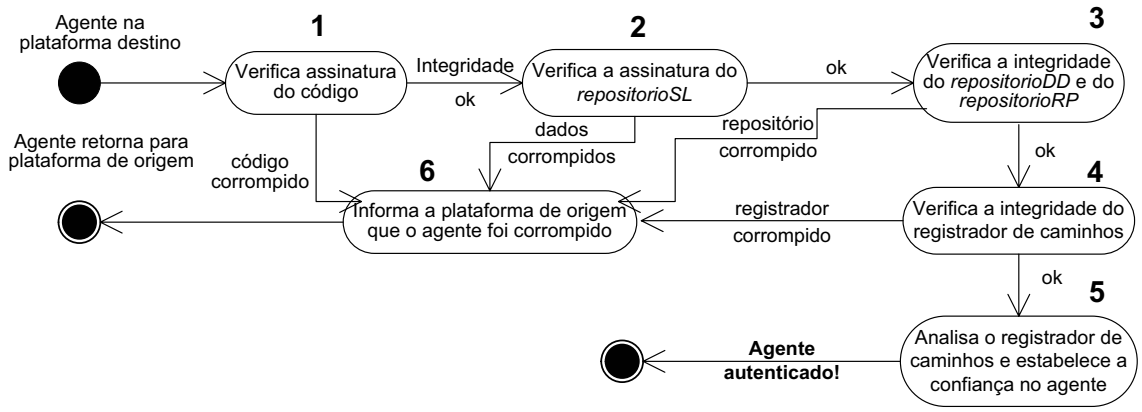


Figura 6.12: Autenticador Multi-hop

entradas do registrador, conforme descrito na seção 6.3.5 (**passo 4**).

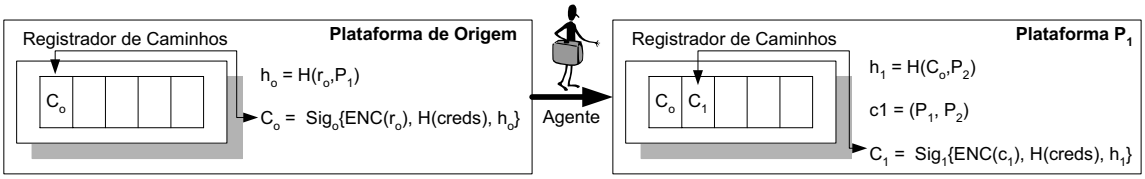


Figura 6.13: Registrador de Caminhos

Vale lembrar que para analisar o histórico do caminho do agente (**passo 5**), a plataforma que está sendo visitada pelo agente deve: (1) analisar as entradas do registrador de objetos, procurando por algumas inconsistência (por exemplo, se o agente não foi desviado); (2) revisar a lista das plataformas visitadas, verificando se estas estão associadas a alguma federação indicada na lista de federações; (3) e, com base na política de segurança definida pelo administrador da plataforma, estabelecer o nível de confiança no agente.

6.5 Controle Social da Reputação das Plataformas

Com o objetivo de tratar as ocorrências de ataques contra os agentes móveis, um mecanismo de controle social, baseado em reputação que visa identificar plataformas maliciosas que integram uma determinada federação (ou grupo), foi definido e integrado ao esquema de segurança proposto. Entretanto, esta abordagem tem um grave problema: a possibilidade de ataques contra a reputação, em que uma plataforma maliciosa forja provas de uma violação visando comprometer a reputação de uma outra plataforma.

A Figura 6.14 ilustra o protocolo definido para identificação de plataformas maliciosas para o controle social proposto. A análise e a definição de reputações das plataformas estão baseadas nas premissas do $MASS_{ma}$ apresentadas na seção 6.3.1.

Conforme apresentado na seção 6.3.3, o mecanismo de controle social trabalha com duas listas que indicam as reputações das plataformas de agentes pertencentes a uma federação

SPKI, a saber, *lista negra* (identificando as plataformas maliciosas) e *lista vermelha* (identificando níveis de reputação das plataformas confiáveis). As reputações indicadas nas listas vermelhas são classificadas de acordo com a quantidade de ocorrências de violações de segurança notificadas por uma plataforma coordenadora do protocolo de identificação. De acordo com o número de notificações de ataques que são atribuídas a uma plataforma, as reputações podem ser quantificadas como no exemplo, a seguir:

Reputação	**Número**	**de Ocorrências**
Plataforma Maliciosa	$\Rightarrow$	4
Plataforma Fortemente Suspeita	$\Rightarrow$	3
Plataforma Moderadamente Suspeita,	$\Rightarrow$	2
Plataforma Levemente Suspeita	$\Rightarrow$	1
Plataforma Totalmente Confiável	$\Rightarrow$	0

Considerando a Figura 6.14, tem-se que um *agenteA*, criado e iniciado na plataforma P_1, com a *QoP* indicada na figura, percorre as plataformas P_2, P_3 e P_4. A plataforma P_4 detecta que o *RepositorioSL* foi corrompido e, por isso, suspende a execução do agente e o manda de volta, com o *RepositorioSL*, para a plataforma de origem para que esta aplique o protocolo de identificação de plataforma maliciosa. P_3 passa a ser suspeita da violação, porém a P_4 também é considerada suspeita, pois esta pode estar tentando denegrir a reputação da plataforma P_3. A plataforma de origem, que é a coordenadora do protocolo de identificação, inicia então o algoritmo de pesquisa de reputações descrito pelos passos, a seguir:

1. De posse do agente, a coordenadora do protocolo de identificação verifica a integridade de todos os repositórios de dados e do registrador de caminhos procurando confirmar a violação e quais plataformas foram visitadas por este agente.

2. Quando a violação é confirmada, esta envia uma mensagem para os gerentes das federações das plataformas que o agente corrompido visitou, solicitando informações sobre a reputação das plataformas suspeitas;

3. Todos os gerentes de federação devem responder a esta notificação indicando as listas de reputações publicadas (listas negras e vermelhas).

4. Com as listas recebidas, a coordenadora deve aplicar a política de controle social para saber como deve proceder.

5. Caso a reputação de uma das plataformas seja considerada **fortemente suspeita**, esta deve enviar uma notificação para o gerente da federação desta plataforma, contendo informações sobre a violação de segurança e a identidade da plataforma considerada suspeita para que esta seja colocada na *lista negra* de reputação e desligada dessa federação. Caso a reputação seja considerada **moderadamente ou fracamente suspeita**,

esta deve enviar uma notificação para o gerente da federação para que a reputação desta plataforma seja atualizada na *lista vermelha* de reputações.

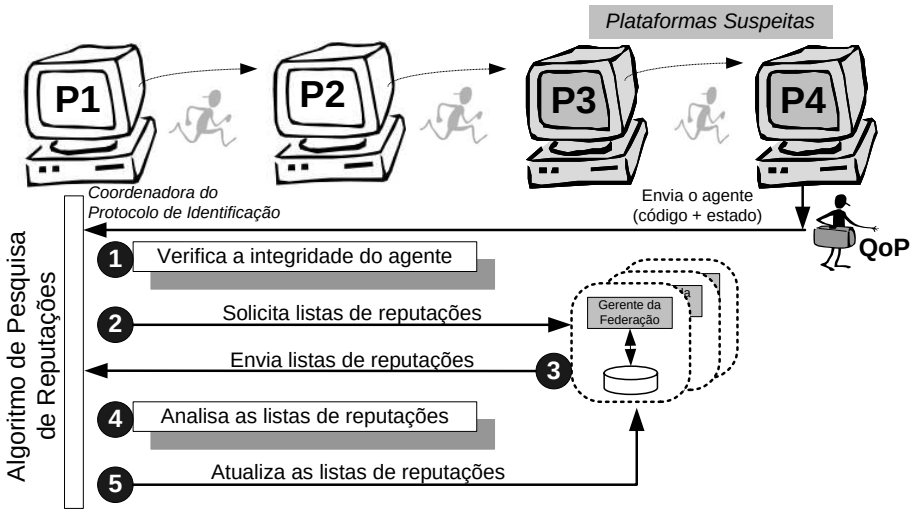


Figura 6.14: Identificação de Plataformas Maliciosas e Suspeitas

6.6 Trabalhos Relacionados

Analisando as plataformas de agentes móveis acadêmicas e comerciais encontradas na literatura, observa-se que a segurança dos agentes móveis contra plataformas maliciosas ainda é um problema não resolvido nestes sistemas. Dentre sete plataformas analisadas[9], somente duas plataformas, SOMA (Corradi *et al.*, 1999a) e Ajanta (Karnik, 1998), preocupam-se em prover mecanismos. Ambas procuram detectar os ataques provenientes de plataformas maliciosas, mas não evitar os complexos ataques de plataformas maliciosas. Para garantir a integridade do código do agente, ambas utilizam técnicas de assinatura digital. Nas duas plataformas, os dados não mutáveis do agente são protegidos através de assinatura digital. Porém, somente a plataforma Ajanta, com o mecanismo Contêiner Somente-Leitura, é que se preocupa em associar as credenciais do agente a estes dados, criando assim uma ligação entre os dados somente-leitura e o agente. O repositório de dados somente-Leitura (*repositorioSL*) proposto neste trabalho está baseado no mecanismo da plataforma Ajanta. A diferença está em como as credenciais do agente são definidas e ligadas aos dados. No $MASS_{ma}$, o objeto *Credenciais*, que será carregado dentro do *repositorioSL*, é construído de forma a distinguir instâncias de um mesmo agente, além disso, o *hash* dos dados do *repositorioSL* fazem parte também deste objeto. Desta forma, uma ligação a prova de modificação entre a instância de um agente e o *repositorioSL* é construída, sendo possível detectar quando uma plataforma maliciosa tenta reutilizar as credencias de um agente.

[9]Foram analisadas as plataformas comerciais *Aglets*, *Concordia*, *Grasshopper*, *Voyager* e as plataformas acadêmicas *Mole*, *SOMA* e *Ajanta*. Detalhes sobre estas plataformas estão descritos no Apêndice B.

Para a proteção dos resultados recolhidos nas plataformas visitadas por um agente, a plataforma *SOMA* utiliza técnicas de detecção, através dos protocolos TTP (*Trusted Third Parties*) e MH (*Multiple-Hop*). O primeiro protocolo necessita de uma terceira parte confiável para encapsular os resultados e, conforme citado anteriormente (seção 6.2.1), esta dependência não é desejável. No protocolo MH, assim como no protocolo de encapsulamento de resultados proposto nesta tese, é responsabilidade da plataforma o encapsulamento dos resultados parciais no estado do agente de forma que seja difícil modificar um resultado intermediário sem modificar também todos os resultados parciais posteriores. Para isto, uma relação de encadeamento entre os resultados necessita ser criptograficamente estabelecida. A plataforma *Ajanta*, através do mecanismo Contêiner Somente para Inclusão, também usa a mesma abordagem de encapsulamento de resultados baseada em relações de encadeamento para garantir a integridade dos resultados parciais. Outro trabalho que segue essa abordagem são os protocolos da Família KAG (Karjoth *et al.*, 1998). Conforme foi observado na seção 6.2.1, a grande diferença entre esses mecanismos está na forma como criptograficamente essas relações de encadeamento são construídas e como estes resultados são anexados ao agente. Nos protocolos da plataforma SOMA e Ajanta, a integridade dos resultados parciais encapsulados não pode ser verificada publicamente. Além disso, a propriedade de integridade em avanço não é garantida por estes protocolos, ou seja, não se pode garantir que quando um resultado parcial for modificado, os resultados inseridos anteriormente permaneçam íntegros. Esses trabalhos relacionados possuem algumas limitações quando uma plataforma é visitada mais de uma vez ou quando os resultados parciais de um agente não malicioso são indevidamente anexados a um agente malicioso. Nestes casos, os protocolos possuem resistência fraca a inserção de resultados falsos e a remoção total de resultados. Tanto os mecanismos da plataforma SOMA quanto os da plataforma Ajanta foram implementados, porém os protocolos da Família KAG não foram implementados. A Tabela B.1 visa comparar algumas características destes mecanismos com os protocolos propostos neste trabalho.

Características e Propriedades de Segurança	MH SOMA	Contêiner Ajanta	P1 KAG	P2 KAG	$MASS_{ma}$ PA	$MASS_{ma}$ PB	$MASS_{ma}$ PC
Verificável publicamente	não	não	sim	não	sim	sim	não
Integridade em avanço[a]	não	não	sim	sim	sim	sim	sim
Confidencialidade	não	sim	sim	sim	sim	não	sim
Resistência forte a inserção de resultados (redundância criptog.)	não	não	não	não	sim	sim	sim
Cifragem simétrica dos dados e cifragem assimétrica das chaves	não	não	não	não	sim	não	sim
Resistência a remoção total de resultados parciais	não	não	não	não	não	não	não

[a] Quando um resultado parcial é modifi cado, somente este resultado e os posteriormente inseridos estão comprometidos.

Tabela 6.5: Comparação dos Mecanismos de Encapsulamento de Resultados

Nos protocolos para inserção de dados no *repositorioRP* suportados no $MASS_{ma}$, as principais características que contribuem para um melhor resultado, quando comparado com os citados acima, são o esquema híbrido de cifragem e a redundância criptográfica adicionada na computação dos resultados parciais protegidos, (*RPs*) que liga o identificador único da instância do agente a cada resultado parcial recolhido nas plataformas visitadas (redundância criptográfica). Esta última característica garante ao esquema uma resistência forte a inserção de resultados parciais. Quanto à propriedade de resistência a remoção total de resultados parciais, quando uma plataforma maliciosa é visitada duas vezes ou quando um conluio de plataformas é constituído, nenhum dos protocolos suportados no $MASS_{ma}$ garante resistência forte à remoção. Porém, quando a seqüencia das plataformas que inseriram dados no *repositorioRP* é comparada com a seqüencia de plataformas visitadas registradas no objeto *registrador de caminhos*, é possível constatar algumas inconsistências e detectar a remoção dos dados inseridos entre as duas visitas. Conforme será apresentado no próximo capítulo, todos os protocolos para inserção de dados no *repositorioRP* foram implementados e testes de desempenho foram realizados, visando medir a degradação provocada por cada protocolo.

Os trabalhos citados acima não oferecem mecanismos que automatizem a verificação da integridade de um agente quando este for recebido em uma plataforma. No $MASS_{ma}$, o autenticador *multi-hop* tem esse papel. Através desse mecanismo, é possível que a qualidade de proteção (*QoP*), atribuída a um agente pelo seu proprietário, seja garantida pelas plataformas visitadas pelo agente. Outra importante contribuição do $MASS_{ma}$ é a técnica de prevenção que evita que um agente móvel seja enviado para uma plataforma considerada não-confiável. Com esta técnica, é possível auxiliar um agente móvel no estabelecimento da confiança em uma plataforma, tendo como base a reputação desta plataforma dentro de um conjunto de plataformas pertencentes a federações de serviços. Conforme apresentado em Rasmusson e Jansson (1996), mecanismos de controle social podem ser combinados a esquemas de segurança, visando definir um comportamento correto que será imposto aos participantes de um grupo. Usufruindo da infra-estrutura das federações de serviço SPKI e visando tratar as ocorrências de ataques contra agentes móveis, um mecanismo de controle social baseado em reputação é suportado no $MASS_{ma}$.

6.7 Conclusões do Capítulo

As limitações das técnicas propostas para tratar o problema das plataformas maliciosas, apontadas nas seções 6.2 e 6.6, comprovam que muitas dessas técnicas ainda não são consideradas práticas e flexíveis e muitas não apresentam resultados satisfatórios que garantam a segurança para os agentes móveis. Tais limitações são ainda maiores quando se leva em conta aplicações em sistemas distribuídos de larga escala. Procurando contornar algumas das limitações apontadas, um esquema de segurança — $MASS_{ma}$— que visa minimizar os riscos a que os agentes móveis *multi-hop* estão suscetíveis, foi proposto. Este esquema oferece ao

proprietário de um agente um conjunto de mecanismos, baseado em repositórios seguros de dados e assinatura digital, que visa construir um agente móvel protegido, conforme as características e exigências de segurança de uma dada aplicação. Além disso, o esquema proposto auxilia as plataformas de agentes na detecção de violações de segurança provocadas por plataformas maliciosas e no processo de identificação e punição dessas plataformas.

Do ponto de vista da proteção de agentes móveis contra plataformas maliciosas, o esquema de segurança proposto neste capítulo atende os seguintes objetivos de segurança:

- detecta quando uma plataforma maliciosa viola a integridade do código de um agente móvel;
- detecta quando uma plataforma maliciosa viola a integridade dos dados somente-leitura carregados pelo agente;
- detecta a violação da integridade dos resultados parciais coletados e carregados por um agente;
- previne a revelação não autorizada dos resultados parciais coletados e carregados por um agente;
- previne a revelação não autorizada de dados direcionados a plataformas pré-definidas;
- aplica um controle social baseado em reputação que visa fiscalizar e punir as plataformas de agentes móveis consideradas suspeitas de violações pertencentes a uma federação SPKI.

A implementação do protótipo que engloba o $MASS_{ma}$ e o $MASS_{ap}$ (apresentado no Capítulo 5), que constitui a base de discussão dos resultados do esquema de segurança para proteção de sistemas baseados em agentes móveis, será discutida no próximo capítulo.

Capítulo 7

Implementação e Resultados Obtidos

7.1 Introdução

O esquema de segurança proposto no Capítulo 4 (Figura 4.8) e detalhado nos Capítulos 5 e 6 — *MASS*— foi projetado com o intuito de oferecer um conjunto de mecanismos de prevenção e de detecção que podem ser combinados de forma a prover uma segurança mais efetiva às aplicações baseadas em agentes móveis, de acordo com as características e política de segurança de cada aplicação. Durante a implementação do esquema, teve-se a preocupação de contrapor os objetivos de segurança de cada mecanismo com aspectos de escalabilidade e desempenho, visando minimizar os impactos do uso desses mecanismos em aplicações distribuídas. Além disso, a portabilidade e a interoperabilidade foram propriedades que nortearam a definição dos mecanismos e das escolhas tecnológicas do protótipo de implementação do *MASS*, conforme descrito neste capítulo.

De forma a avaliar os mecanismos que compõem o protótipo de implementação do *MASS*, uma análise do desempenho é descrita neste capítulo. Feita a análise, diretrizes para seleção dos mecanismos mais adequados a uma dada aplicação são definidas e empregadas para escolha das técnicas de proteção a serem utilizadas em uma aplicação distribuída. Por fim, a integração do protótipo a uma aplicação distribuída — Busca e Seleção de Parceiros para Formação de Empresas Virtuais — é discutida com objetivo de analisar a potencialidade do esquema de segurança proposto.

Os resultados obtidos com o esquema de segurança e sua implementação foram absorvidos em dois projetos de pesquisa: o projeto Cadeias de Confiança (CNPq/PROTEM - Conteúdos Digitais)[1] e o projeto Instituto Fábrica do Milênio - IFM (MCT/CNPq)[2], ambos desenvolvidos no Departamento de Automação e Sistemas da UFSC.

[1] http://www.das.ufsc.br/seguranca
[2] http://www.ifm.org

7.2 Protótipo de Implementação do *MASS*

Um protótipo, envolvendo o esquema de segurança — *MASS*, foi definido e implementado visando comprovar a sua flexibilidade e a viabilidade de sua utilização em aplicações distribuídas que embutem a noção de mobilidade de código. As escolhas tecnológicas necessárias para compor a arquitetura do protótipo, ilustrada na Figura 7.1, têm como base o uso de padrões abertos e componentes de prateleiras (*Commercial Off-the-Shelf -COTS*) para que os requisitos de portabilidade, escalabilidade e interoperabilidade sejam preservados. Outros padrões no contexto da Internet também foram considerados. As camadas que compõem a arquitetura do protótipo serão apresentadas, a seguir.

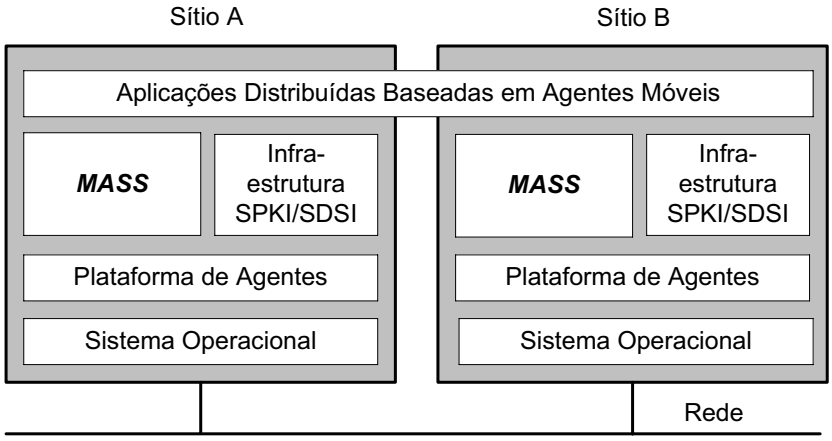


Figura 7.1: Arquitetura do Protótipo

7.2.1 Plataforma de Agentes

Para compor a camada do ambiente computacional que suporta os agentes móveis, foi adotada a **plataforma Aglets** da IBM (1996)[3], primeiramente por usar o Java como linguagem de código móvel e também por ser uma iniciativa de código aberto, permitindo assim que extensões e modificações necessárias para integração do esquema de segurança fossem possíveis. Conforme apresentado nos Capítulos 3 e 4, a linguagem Java, além de ser considerada um padrão de fato para a programação de aplicações distribuídas, possui diversas propriedades e mecanismos que a tornam uma boa linguagem para programação de agentes móveis [4].

O Kit de Desenvolvimento de Software Aglets (ASDK), na versão 2.0.2, fornece ainda outras boas características que foram importantes para sua escolha como plataforma de agentes, a saber:

[3]Uma visão geral da plataforma Aglets e dos mecanismos de segurança suportados nesta plataforma são descritos no Apêndice B.

[4]Os mecanismos que contribuem com o modelo de segurança bem como com os mecanismos de segurança suportados na plataforma Java estão descritos no Apêndice A.

- garante a mobilidade de código, de dados e de informações de estado;
- possui um ambiente computacional padrão, chamado Tahiti, que suporta criação, clonagem, execução e envio de agentes e oferece uma interface gráfica amigável (*GUI*);
- permite interação entre agentes móveis através da troca de mensagens síncronas ou assíncronas;
- possui uma API de comunicação derivada do padrão MAF;
- suporta o ATP (*Agent Transfer Protocol*) e o Java RMI (*Remote Method Invocation*) como infra-estruturas de comunicação;

Um estudo detalhado da plataforma Aglets-2.0.2 foi realizado no contexto do projeto IFM, visando conhecer sua arquitetura, seu modelo de agentes, os mecanismos de segurança suportados e a JAVA Aglet API (*Application Programming Interface*) — *J-AAPI* — para melhor integrar o esquema de segurança proposto[5] .

7.2.2 Infra-estrutura SPKI/SDSI

A infra-estrutura SPKI/SDSI, integrante da arquitetura do protótipo, é responsável pela criação, emissão e gerenciamento dos certificados SPKI/SDSI, pela verificação das cadeias de certificados de autorização e por oferecer o suporte necessário para as federações SPKI. Uma biblioteca de códigos Java que implementa o SPKI/SDSI, chamada JSDSI2.0 (Morcos, 1998), e que provê algumas dessas funcionalidades, foi utilizada no protótipo. O JSDSI2.0 possui uma interface gráfica que permite criação de pares de chaves, gerenciamento de certificados e a verificação de cadeias de certificados. Para fornecer as funcionalidades de criptografia, esta biblioteca usa o *Cryptix32* [6], porém, somente chaves RSA são suportadas.

A biblioteca JSDSI foi estendida para que a infra-estrutura SPKI/SDSI se tornasse mais flexível e eficiente para ser usada pelas plataformas de agentes e para agilizar a busca de certificados SPKI. Toda a habilidade de se trabalhar com objetos SPKI/SDSI no protótipo é garantida através da classe `SPKIResolver` e esta foi desenvolvida dentro do projeto Cadeias de Confiança.

Como os objetos SPKI/SDSI são descritos em *S-expressions* (Rivest, 1997), estes podem ser armazenados no formato ASCII (*American Standard Code for Information Interchange*), podendo estar em um arquivo texto ou até mesmo em um banco de dados relacional. A ferramenta presente na biblioteca JSDSI2.0 utiliza como repositório um simples arquivo texto.

[5]Como resultado deste estudo, um relatório técnico que descreve em profundidade a plataforma Aglets foi redigido e encontra-se disponível na página do Grupo de Estudos de Agentes Móveis e Segurança — http://www.das.ufsc.br/AgenteSeg/relatorios.

[6]http://www.cryptix.org/products/cryptix31/index.html

Para facilitar o gerenciamento dos objetos SPKI/SDSI e agilizar a busca de certificados, a biblioteca *parserSxxS* (Mello, 2003) foi implementada no contexto do projeto Cadeias de Confiança, possibilitando que todos os objetos suportados pela biblioteca JSDSI possam ser convertidos para documentos XML (*eXtensible Markup Language*)[7] e vice-versa, tendo como base o DTD (*Document Type Definition*), especificado em (Orri e Mas, 2001). Assim, a biblioteca *parserSxxS* permite a completa adoção de documentos XML como uma forma para armazenar objetos SDSI ao invés de *S-expressions*, tornando mais ágil a manipulação destes documentos (Mello, 2003).

Nas Federações SPKI/SDSI, implementadas no projeto Cadeias de Confiança, há dois tipos de repositórios de certificados: os repositórios locais, os quais situam-se juntos de cada membro da federação; e os repositórios globais, os quais situam-se juntos dos gerentes das federações. Foi com base nos documentos XML que foram implementados os repositórios de certificados do protótipo — pacote `repository.local` e pacote `repository.global` (Mello, 2003). O pacote `repository.local` consiste na implementação de uma estrutura de arquivos, que representa os documentos XML armazenados no repositório, e ainda os mecanismos para manutenção desta base de dados como a inclusão, a modificação, a remoção de arquivos, bem como a busca por alguma informação dentro destes arquivos[8]. Para o repositório global foi utilizado o *Apache Xindice*[9], um banco de dados que armazena documentos XML de forma nativa. O *Xindice* utiliza o *XPath* (Clark e DeRose, 1999) como linguagem para pesquisa de documentos (*query language*), assim o pacote `repository.global` consiste basicamente da implementação de mecanismos que propiciem fazer consultas, inclusões, remoções e modificações rápidas e simples na base de dados, implementada pelo *Xindice*.

7.2.3 Esquema de Segurança — *MASS*

Esta camada do protótipo é composta por: interfaces gráficas que auxiliam a configuração e inicialização do esquema de segurança; uma biblioteca de classes, chamada **AgentSec**, que auxilia o programador de um agente na construção de agentes móveis protegidos; objetos que estendem as funcionalidades da plataforma Aglets para suportar o esquema de segurança; objetos que implementam o protocolo de autenticação mútua entre plataformas; o esquema para geração de domínios de proteção e, o autenticador *multi-hop*. As próximas seções detalham a implementação do esquema de segurança e sua integração à plataforma Aglets.

As ferramentas de programação utilizadas na implementação do esquema de segurança

[7] O XML é um padrão da W3C (*World Wide Web Consortium*) cujo principal objetivo é o intercâmbio de conteúdos digitais (Bray *et al.*, 1998). O XML traz flexibilidade às aplicações, permitindo buscas mais eficientes e distribuição dos dados de forma mais ampla e escalável. Diversas aplicações já utilizam este padrão o que o torna um padrão de fato no contexto da Internet.

[8] O parser SAX —*Simple API for XML* (http://www.saxproject.org/), presente no J2SE 1.4, foi utilizado na implementação do repositório local.

[9] http://xml.apache.org/xindice

foram: **J2SE v1.4.2 SDK** (plataforma Java 2)[10], **Eclipse 2.1** (ambiente de desenvolvimento Java)[11], **Xindice 1.0** (banco de dados XML nativo), **iSaSiLk 3.06 toolkit** [12] (implementação do protocolo SSL), **IAIK-JCE 3.0.3 toolkit**[13](implementação da API JCE do Java) e **Together 6.0.2** (ferramenta para desenvolvimento de software utilizando UML).

7.3 Configuração do Esquema de Segurança

Na plataforma Aglets, o administrador configura o ambiente computacional Tahiti editando um arquivo de configuração de propriedades (*aglets.props*). Para facilitar a configuração do esquema de segurança e do próprio Tahiti, uma interface gráfica foi implementada e integrada à plataforma Aglets (ver Figura 7.2). A partir desta interface, o administrador pode definir quais serão os mecanismos de segurança usados para proteger as plataformas de agentes. Após configurada a plataforma, o arquivo de propriedades (*aglets.props*) é automaticamente gerado pela interface. Quando o ambiente Tahiti é iniciado com as propriedades definidas pelo administrador, caso o protocolo de autenticação mútua de plataformas esteja selecionado, o agente estacionário `Authenticator` é iniciado junto com a plataforma e, caso o mecanismo de autenticação de agentes seja requerido pelo administrador, o agente estacionário `SecurityInterceptor` é também iniciado. Estes agentes estacionários são os principais responsáveis por implantar os mecanismos de estabelecimento de canal seguro e autenticação de agentes, respectivamente, em cada plataforma Aglets. Mais detalhes sobre estes agentes serão vistos nas próximas seções.

Visando auxiliar o administrador da plataforma de agentes na configuração da política de controle de acesso a ser aplicada pela plataforma, uma aplicação, chamada `SPKIPolicyTool`, foi desenvolvida. Esta aplicação é apresentada em detalhes na seção 7.7.

Relatórios técnicos que descrevem o funcionamento da interface para configuração do Tahiti e da aplicação `SPKIPolicyTool` estão disponíveis na página do Grupo de Estudos de Agentes Móveis e Segurança do DAS (*AgenteSeg*) http://www.das.ufsc.br/AgenteSeg/relatorios.

7.4 A Criação de Agentes Móveis Protegidos com a Biblioteca *AgentSec*

O *MASS*, através da biblioteca `AgentSec`, oferece ao programador de agentes móveis três repositórios seguros (de dados somente-leitura, de dados direcionados e de resultados parciais) e um objeto registrador de caminhos, que visa manter um histórico das plataformas visitadas pelo agente. Os objetos desta biblioteca foram implementados para serem

[10] http://java.sun.com/j2se/1.4.2/
[11] http://www.eclipse.org/
[12] http://jce.iaik.tugraz.at/products/02_isasilk/index.php
[13] http://jce.iaik.tugraz.at/products/01_jce/index.php

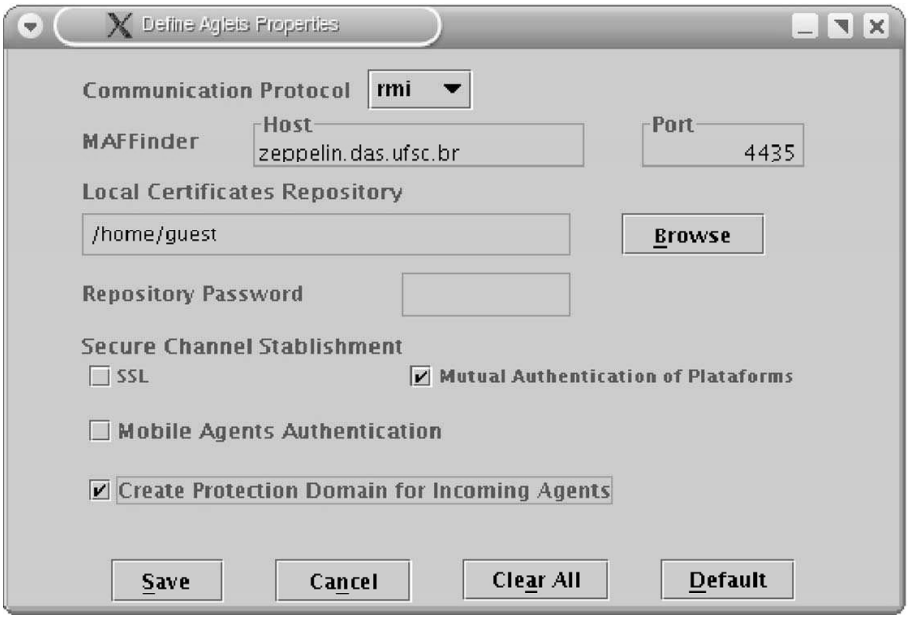


Figura 7.2: Interface para Definição dos Mecanismos de Segurança de uma Plataforma Aglets

independentes de plataforma de agentes. Ou seja, esta biblioteca pode ser ainda facilmente utilizada para criação de agentes móveis que seguem outros modelos de agentes que diferem do adotado pela plataforma Aglets.

Para a seleção dos mecanismos de proteção dos agentes móveis e para auxiliar o uso dos repositórios seguros de dados, o programador de agentes conta com uma interface gráfica que auxilia a definição da qualidade de proteção do agente (*QoP*). Esta interface, ilustrada na Figura 7.3, também ajuda o programador do agente a editar níveis de proteção de acordo com a seleção dos mecanismos de segurança. Após a seleção dos mecanismos na interface, o programador do agente tem como resultado um esqueleto para o código do agente (em um arquivo .java). De posse dessa estrutura gerada e da biblioteca `AgentSec`, o programador pode facilmente continuar a implementação do agente móvel. A estrutura gerada para o agente, quando a qualidade de proteção é definida conforme ilustrada na Figura 7.3, é apresentada no Apêndice C.

Conforme ilustrado na Figura 7.3, na definição da qualidade de proteção, o programador deve escolher qual protocolo para inserção de resultados parciais será utilizado e ainda indicar se a verificação deste repositório será só na plataforma de origem (*home platform*) ou em qualquer plataforma visitada. Lembrando que no caso do protocolo C, o repositório poderá ser verificado somente na plataforma de origem.

A Figura 7.4 ilustra a visão das classes implementadas que compõem a biblioteca `agentSec`. O *repositorioSL*, o *repositorioDD*, *repositorioRP* e o registrador de caminhos, implementados nas classes `ROReposito ry`, `DDReposito ry`, `PRReposito ry` e `PathRegister`, respectivamente, quando selecionados, são criados e definidos no método `oncreation` por qualquer agente que estenda a classe abstrata `com.ibm.aglet.Aglet`. Vale ressaltar que, durante

Figura 7.3: Interface para Definição da Qualidade de Proteção para um Agente Móvel

o processo de criação de um objeto `RORepository`, o objeto `Credentials` associado ao agente é criado e armazenado neste repositório. Conforme definido no Capítulo 5, o objeto `QoP` é um atributo do objeto `Credentials`.

Um mecanismo para assinar digitalmente o código dos agentes móveis também está disponível no *MASS*. Como a implementação deste mecanismo está diretamente relacionada com a implementação da Plataforma Aglets, o suporte a este mecanismo não é oferecido através da biblioteca `AgentSec`. A definição do uso ou não deste mecanismo é realizado pelo programador via interface de definição do *QoP*. A plataforma Aglets precisou ser adaptada para opcionalmente usar o mecanismo de assinatura de código, quando o código é carregado com o agente ou quando este é solicitado sob demanda[14].

7.5 Estabelecendo um Canal Seguro

No *MASS*, um canal seguro é estabelecido quando a autenticação mútua entre as plataformas é bem sucedida e quando uma tecnologia de segurança subjacente é utilizada para garantir a integridade e confidencialidade dos dados e de agentes que trafegam por este canal. O protocolo para autenticação mútua de plataformas de agentes foi implementado com o auxílio da biblioteca JSDSI2.0, do objeto `SPKIResolver` e do repositório local de certificados. Esse protocolo foi implementado em um agente móvel (`Authenticator`), que é acionado sempre que uma plataforma de agentes tenta se comunicar com outra plataforma, e estendendo a classe responsável pela comunicação entre as plataformas de agentes

[14]Uma descrição detalhada das adaptações necessárias para que a plataforma Aglets suporte assinatura de código está descrito em um relatório técnico disponível em http://www.das.ufsc.br/agenteSeg/relatorios

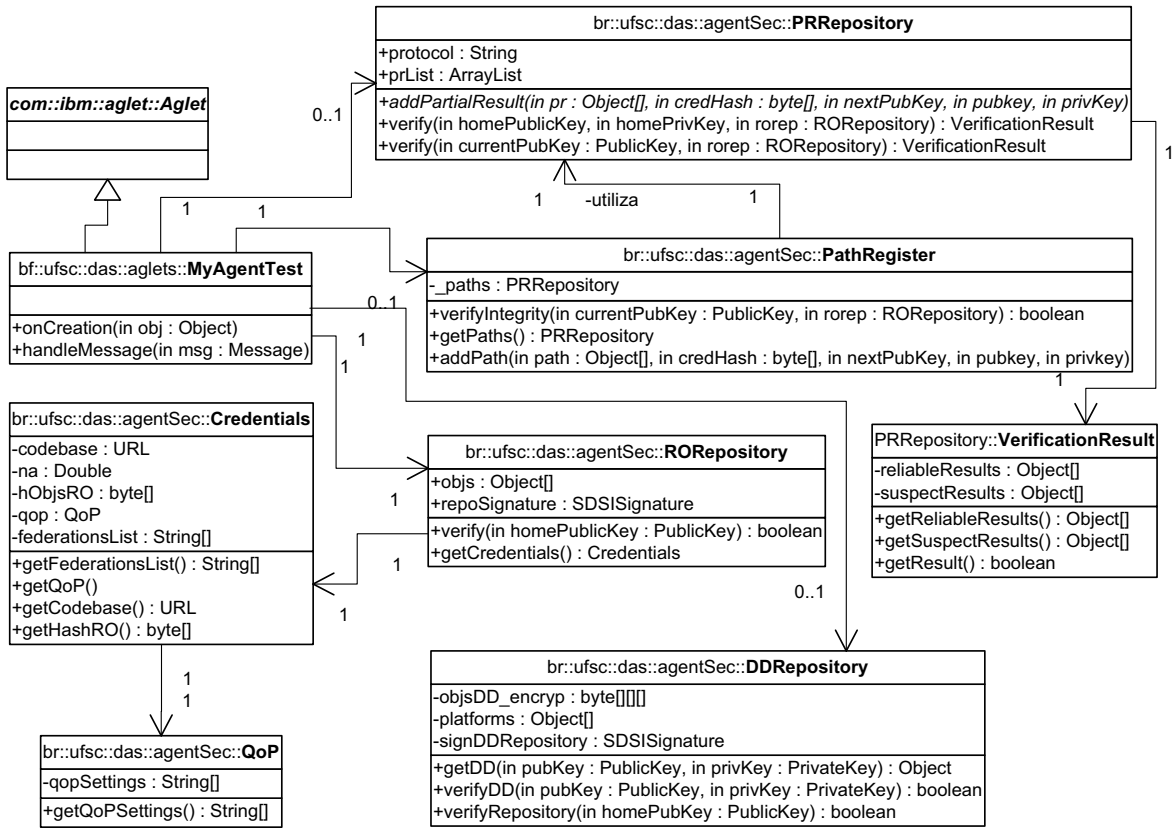


Figura 7.4: Diagrama de Classes da Biblioteca AgentSec

(`com.ibm.rmi.Handler`). Se um canal seguro já está estabelecido, a comunicação ocorre normalmente. Senão, o protocolo se inicia para estabelecer a autenticação mútua entre as plataformas através de troca de mensagens entre a classe `Handler` e o agente `Authenticator` da plataforma-destino (protocolo *challenge-response*). Quando a autenticação mútua é realizada com sucesso, o canal seguro permanece válido para as próximas interações entre as duas plataformas. Em caso negativo, uma janela de erro aparecerá na interface da plataforma-fonte e da plataforma-destino[15].

Para a geração das chaves de sessão, necessárias para garantir a integridade e a confidencialidade do canal de comunicação, utilizou-se, como tecnologia de segurança subjacente, o protocolo SSL, conforme implementado pela ferramenta *iSaSiLk*. Esta interação com o SSL foi implementada e integrada à camada de comunicação da plataforma Aglets. Desta forma, a plataforma Aglets foi adaptada para usar, opcionalmente, o Java RMI sobre o SSL em todas as comunicações (entre plataformas de agentes e com o Serviço de Nomes — `MAFFinder`)[16]. Os certificados SSL necessários foram gerados a partir dos certificados SPKI das plataformas de agentes móveis, utilizados no protocolo de autenticação mútua.

[15]Um relatório técnico foi redigido documentando todas as extensões realizadas na plataforma Aglets para dar o suporte ao protocolo de autenticação mútua, bem como descrevendo o agente *Authenticator*. Este relatório está disponível em http://www.das.ufsc.br/gtAgentSeg/relatorios.

[16]Detalhes sobre a integração do protocolo SSL à plataforma Aglets podem ser obtidos em http://www.das.ufsc.br/gtAgentSeg/relatorios

7.6 Autenticação de Agentes Móveis

O autenticador de agentes *multi-hop* proposto visa não só auxiliar uma plataforma no estabelecimento da confiança nos agentes que esta recebe, mas também detectar possíveis violações de integridade, ajudando ainda no processo de identificação de plataformas maliciosas. Devido às particularidades da plataforma Aglets, mostrou-se mais adequado criar um agente estacionário que intercepta o recebimento de agentes em uma plataforma e que implementa o algoritmo do autenticador *multi-hop* — o agente `SecurityInterceptor`. Este agente é iniciado em cada plataforma que deseja implantar a autenticação de agentes móveis e fica à espera de mensagens que indicam a chegada de um agente para iniciar a interceptação e o processo de autenticação deste agente. O `SecurityInterceptor` executa ainda outras funções essenciais para auditoria da plataforma, como a criação de um arquivo de *log* para manutenção de um registro de suas atividades e de todas as interações e verificações com os agentes visitantes.

Quando um agente é recebido em uma plataforma, através do método `receive_agent` da classe `com.ibm.aglets.AgletContextImpl`, o carregador de classes da plataforma Aglets é acionado (`com::ibm::aglets::tahiti::AgletClassLoader`). Este carregador precisou ser estendido, pois somente este poderia verificar a assinatura do código do agente visitante, antes que uma *thread* fosse criada para o agente. Após verificar a assinatura do código, o carregador de classes deve informar ao `SecurityInterceptor` o resultado desta verificação. Ainda sobre o método `receive_agent`, após o agente ter sido carregado, uma mensagem é enviada para o agente `SecurityInterceptor` para que este continue a execução do algoritmo do autenticador *multi-hop*, verificando a integridade dos repositórios de dados e do registrador de caminhos (ver Figura 7.5). Todas as entradas do registrador de caminhos são verificadas, porém não é analisado se as plataformas visitadas são membros ou não das federações indicadas pelo proprietário do agente. Além disso, somente dois níveis de confiança foram definidos: confiável ou não confiável. Um agente é considerado confiável se todos os passos anteriores forem bem sucedidos. Um diagrama de atividades do `SecurityInterceptor` é ilustrado na Figura 7.5.

7.7 Geração de Domínios de Proteção para Execução de Agentes Móveis

Os procedimentos para geração dos domínios de proteção na plataforma Java, com base nos atributos de privilégios do agente — cadeias de autorização SPKI— foram implementados conforme definido no Capítulo 5 (seção 5.3.4) e integrados na plataforma Aglets. Para esta integração, algumas classes foram estendidas e outras foram criadas[17]. Quando um

[17]Um relatório técnico descreve a implementação dos procedimentos para geração de domínios de proteção e de sua integração à plataforma Aglets (http://www.das.ufsc.br/AgenteSeg/relatorios)

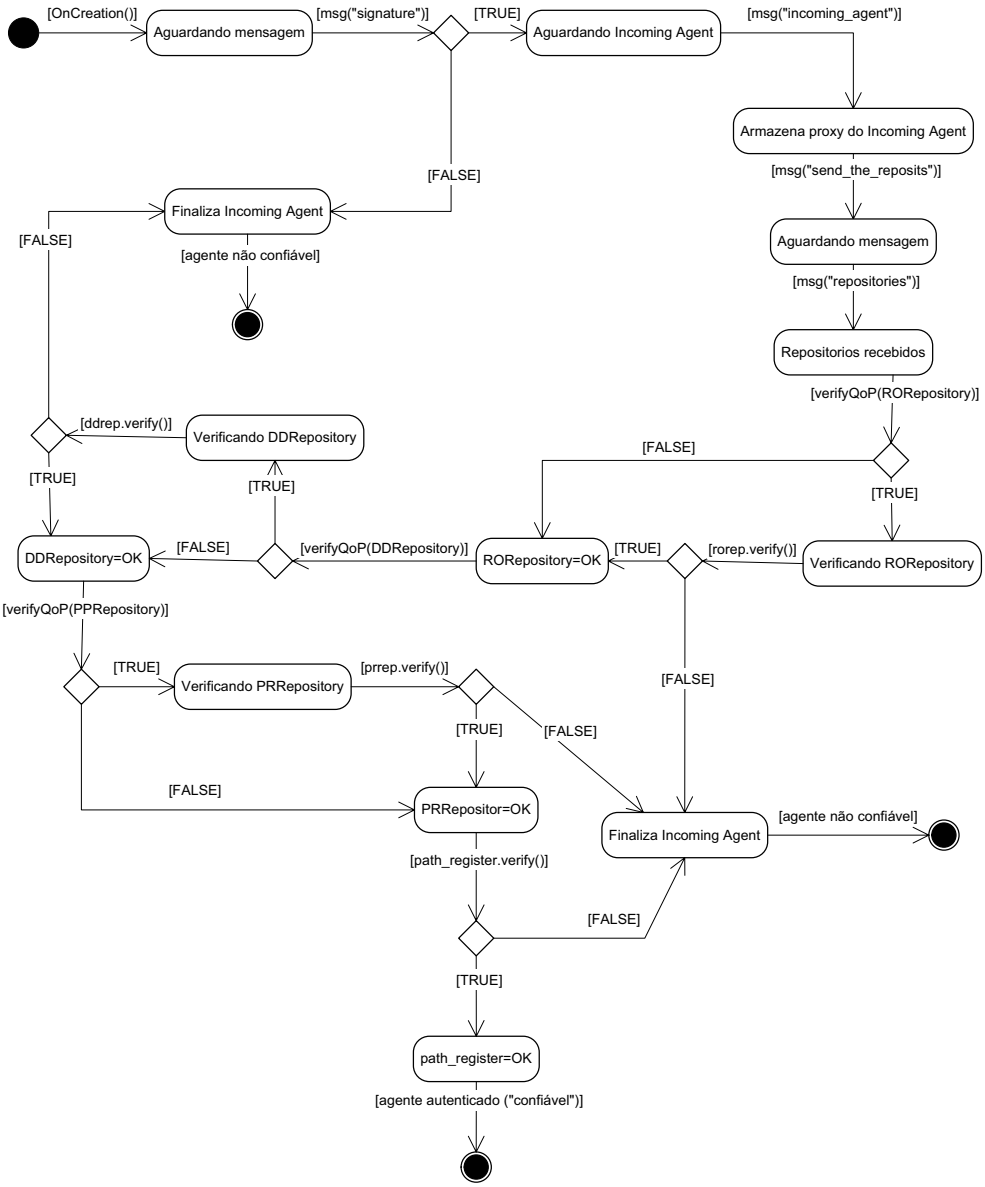


Figura 7.5: Diagrama de Atividades do *SecutityInterceptor*

agente é criado em uma plataforma de agentes, seu proprietário deve definir os atributos de privilégios que este agente carregará consigo — suas cadeias de autorizações inicias. Para isto, sempre que um agente for criado, seu proprietário pode contar com uma interface gráfica que auxilia a associação das cadeias de autorização com cada agente.

Quando um agente é criado, as cadeias de autorização definidas pelo proprietário são anexadas ao agente através do método `createAglet` da classe `com::ibm::aglets: AgletContextImpl`. Quando este agente for disparado, a classe `br::ufsc::das::DPJava:: SPKIAgletWriter` é acionada para realizar a serialização do agente, juntamente com suas cadeias de autorização. Quando o agente é recebido na plataforma-destino, o método `receiveAglet` da classe `AgletContextImpl` chama a classe `br::ufsc::das:: DPJava::SPKIAgletReader` para executar a desarilização do agente, bem como extrair as cadeias de autorização do agente e

repassá-las para o carregador de classes da plataforma Aglets — `AgletClassLoader` A partir dos atributos de privilégios concedidos ao agente, expressos nas cadeias de autorização, e da política de segurança da plataforma de agentes corrente, o `AgletClassLoader` define quais permissões são atribuídas ao domínio de proteção.

A política para um ambiente de aplicação da linguagem Java é representada por um objeto `Policy`. Mais especificamente, por subclasses que implementam os métodos abstratos da classe `Policy`. Para que a política de segurança fosse baseada em cadeias de autorização SPKI, a classe `br::ufsc::das::DPJava::SPKIPolicy` foi implementada e integrada ao ambiente da plataforma Aglets . Quando o método `SPKIPolicy.getPermissions` é chamado pelo `AgletClassLoader`, este mapeia os atributos de privilégios contidos nas cadeias de autorização, que ligam o proprietário do agente à plataforma corrente, em permissões Java e então cria o domínio de proteção para o agente. Um arquivo de configuração de política, conforme apresentado, a seguir, é utilizado na execução deste mapeamento.

Para criar e gerenciar a política de segurança a ser aplicada pela plataforma Aglets (máquina Java), especificando quais permissões são concedidas a um dado atributo de privilégio (um papel), uma aplicação Java, chamada ***SPKIPolicyTool***, foi desenvolvida. Esta aplicação, que visa auxiliar o administrador da plataforma, está baseada na ferramenta *PolicyTool* do Java. Entretanto, nesta aplicação, as permissões Java não estão associadas ao *codebase* do código como no *PolicyTool*, mas sim associadas a atributos de privilégios. Outra diferença dessas ferramentas é que o arquivo de configuração de política gerado com o *SPKIPolicyTool* é um documento XML que segue o formato definido em um DTD (*Data Type Definition*) e não um arquivo texto que segue a sintaxe definida pela Sun como no *PolicyTool*[18]. O DTD auxilia a validação do documento XML, gerado a partir da aplicação *SPKIPolicyTool*, garantindo que sua sintaxe está correta[19]. Um exemplo de uso da aplicação *SPKIPolicyTool* e do documento XML gerado é apresentado nas Figuras 7.6 e 7.7.

7.8 Considerações sobre a Implementação do Protótipo

A Figura 7.8 ilustra todos os mecanismos e procedimentos do *MASS* que foram implementados e perfeitamente integrados à plataforma Aglets. O protótipo que implementa o conceito de teia de federações, adotado no *MASS* e proposto em Santin (2004), ainda não se encontrava totalmente implementado quando o protótipo do *MASS* foi desenvolvido. Como o mecanismo de controle social e a técnica para verificação da reputação das plataformas proposto no *MASS* dependem da infra-estrutura da teia de federações SPKI, estes não foram implementados. Também por isto, a análise mais detalhada do histórico das plataformas

[18] A sintaxe definida pela Sun para o arquivo de configuração de política está disponível em http://java.sun.com/j2se/1.4.2/docs/guide/security/PolicyFiles.html

[19] O DTD para especificação da `SPKIPolicy` é apresentado no Apêndice D.

Figura 7.6: Exemplo do Uso da Aplicação SPKIPolicyTool

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policy SYSTEM "policy.dtd">
<policy>
      <grant>
            <privAttrib>messenger</privAttrib>
            <description>messenger mobile agent</description>
            <permissionCollection>
                  <permission>
                        <java.io.SerializablePermission>
                              <enableSubclassImplementation/>
                        </java.io.SerializablePermission>
                  </permission>
                  <permission>
                        <java.util.PropertyPermission
PropertyPermissionActions="read, write">
                              <any>/home/config/savedAglets.props</any>
                        </java.util.PropertyPermission>
                  </permission>
            </permissionCollection>
      </grant>
</policy>
```

Figura 7.7: Arquivo XML da Política SPKI conforme definida na Figura 7.6

visitadas por um agente, que também dependeria da teia de federação SPKI, não pôde ser realizada.

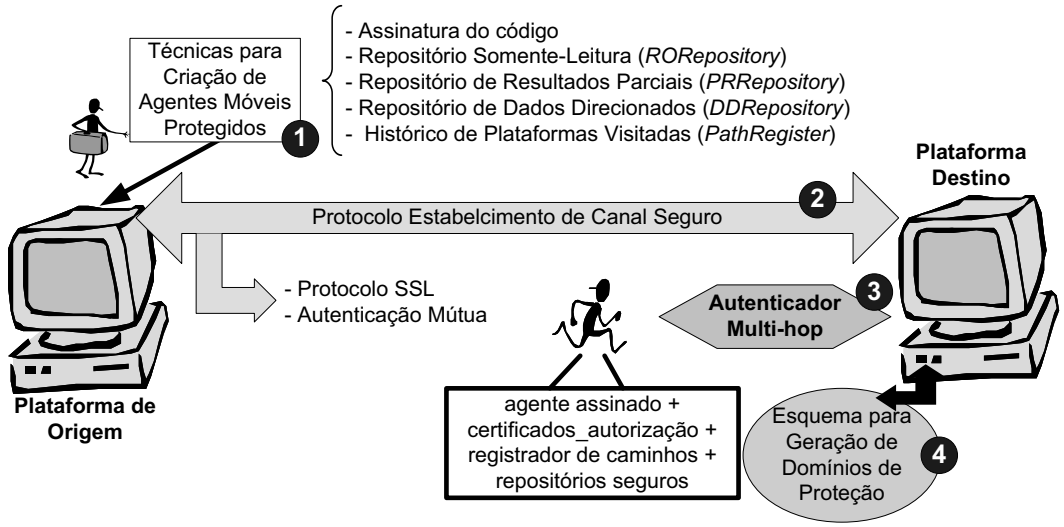


Figura 7.8: Mecanismos e Procedimentos do *MASS* Implementados no Protótipo

Visando melhorar a confiabilidade do protótipo do MASS, durante o seu desenvolvimento, procurou-se analisar a sua corretude através de atividades de verificação e validação (V&V) — testes de *software*[20]. Testar é um método dinâmico para verificação e validação em que o comportamento do sistema é observado pela execução do sistema (Jalote, 1991). Os níveis básicos de testes para tentar detectar diferentes tipos de falhas são: testes de unidade, testes de integração e testes de sistema e de aceitação. A relação das falhas introduzidas em diferentes níveis de testes é apresentada na Figura 7.9. O primeiro nível de teste, chamado

[20] Verificação é o processo que determina se os resultados (produtos) de uma determinada fase do desenvolvimento do *software* preenchem as especificações dos requisitos estabelecidos durante a fase anterior e Validação é o processo que avalia o *software* no final do seu processo de desenvolvimento para garantir a conformidade com os requisitos definidos para esse software (Jalote, 1991).

teste de unidade, é usado para testar a lógica interna dos módulos. Seu objetivo é detectar erros na codificação dos módulos (verificação). No próximo nível, chamado teste de integração, módulos testados são combinados em sub-sistemas para então serem testados. O objetivo aqui é testar o projeto do sistema e então verificar as interações entre os módulos. Os próximos níveis são os testes de sistema e o de aceitação. Nestes níveis o sistema inteiro é testado, validando assim o sistema de acordo com os requisitos definidos para esse.

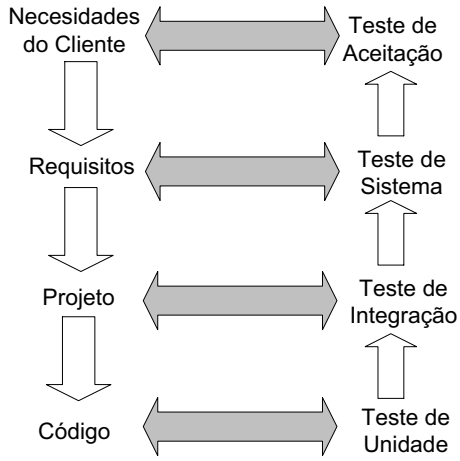


Figura 7.9: Níveis de Testes

Um documento básico para guiar o processo dos testes do protótipo do *MASS* — Plano de Testes — foi redigido visando especificar os níveis de testes, as unidades a serem testadas, os casos de testes (*testcases*), bem como a abordagem dos testes e o cronograma para execução dos testes. Dois tipos de testes foram executados: testes de unidade e testes de sistema. As unidades testadas foram: os objetos responsáveis pelo protocolo de autenticação mútua, os objetos que implementam as técnicas para criação de agentes móveis protegidos e a autenticação de agentes móveis e os objetos responsáveis pelo processo de geração de domínios de proteção. Os testes de unidades seguiram uma abordagem estrutural (teste caixa-branca), tendo como critério a cobertura de ramos (execução de caminhos lógicos dos módulos) e foram aplicados nos agentes estacionários (`Authenticator` e `SecurityInterceptor`), nos repositórios seguros de dados (`RORepository`, `DDRepository` e `PRRepository`), no objeto que representa a política de segurança (`SPKIPolicy`) e nas interfaces gráficas que auxiliam a configuração do esquema de segurança e o seu uso (`SPKIPolicyTool`, `QOPSettings` e `AgletsPropertiesSettings`) [21].

Os testes de sistema realizados seguiram a abordagem funcional (teste caixa-preta) com o objetivo de verificar os requisitos e especificações do protótipo. Os métodos utilizados foram o particionamento de classes de equivalência (utilização de um conjunto de valores válidos e inválidos), a análise de valores limites e de conjuntos de dados especiais e os grafos de causa-efeito para testar combinações de entradas.

[21] Diagramas de atividades UML foram elaborados para todas as unidades testadas.

Todas as faltas e erros detectados nos testes de software foram corrigidos. A documentação dos testes de software do protótipo do *MASS*, que compreende o plano de testes e os relatórios dos testes de unidade e de sistema, está disponível em `http://www.das.ufsc.br/AgenteSeg/testes`. A documentação do código-fonte Java (em arquivos HTML) foi gerada com o utilitário `javadoc` do Java 2 e está disponível em `http://www.das.ufsc.br/AgenteSeg/javadoc`.

7.9 Avaliação de Desempenho

Nesta seção são apresentados alguns resultados de testes não funcionais realizados com o objetivo de avaliar o desempenho do protótipo do *MASS*, ou seja, avaliar o custo adicional introduzido ao integrar mecanismos de segurança (SSL e o protocolo de autenticação mútua) à plataforma Aglets e ao construir agentes móveis que utilizam os repositórios seguros de dados. É importante destacar que os primeiros resultados dos testes de desempenho do protótipo não se mostraram satisfatórios. Diante de degradações elevadas de desempenho, uma revisão e correção da implementação de todo o protótipo foi realizada. Utilizando técnicas de melhoria de desempenho[22], conforme proposto em Haggar (2000), teve-se como resultado um código menor e bem mais rápido de ser executado quando comparado com os primeiros resultados. Os resultados apresentados nesta seção são os do protótipo corrigido.

Os testes foram realizados no LCMI/DAS, usando uma rede local (Fast Ethernet - 100 Mbps) e dois computadores dedicados com configurações idênticas — Pentium IV 2.4 GHz com 512 MB de memória RAM, tendo como sistema operacional GNU/Linux (kernel 2.4.21-199-atlon). Ambos computadores dispunham da versão 1.4.2-04 do *Java 2 Software Development Kit* (J2SDK) para compilação e execução dos aplicativos Java. Para visualizar a migração dos agentes entre as máquinas e com o intuito de facilitar a obtenção das medidas, foi utilizado o ambiente computacional Tahiti, sendo executado a partir da IDE Eclipse 2.1.

Foram criados três cenários de testes: um para avaliar a integração do SSL à plataforma Aglets (RMI sobre SSL), outro para avaliar o uso do protocolo de autenticação mútua de plataformas Agltes e o último para avaliar o uso dos repositórios seguros de dados e o processo de autenticação de agentes móveis. Para os dois primeiros cenários, um agente móvel *two-hop* bumerangue [23], que logo após a sua inicialização salta da sua plataforma de origem para outra plataforma e depois volta, foi construído.

[22] Não foi adotada nenhuma ferramenta de otimização de código, mas sim boas técnicas de programação Java.
[23] Necessário para que não houvesse necessidade de sincronização de relógios das máquinas

7.9.1 Cenário 1: Plataforma Aglets + Protocolo SSL

A primeira medida realizada foi a da latência. No $MASS_{ap}$, quando uma plataforma de agentes se registra no *MAFFinder* (serviço de nomes) ou quando envia um agente ou uma mensagem para uma outra plataforma de agentes, um conjunto de procedimentos para o estabelecimento do contexto de segurança SSL — *handshake* SSL — é executado entre as duas plataformas. Os algoritmos criptográficos (contexto SSL) usados nos testes foram: o *RSA* (chave de 1024 bits) para distribuição de chaves, o *3DES* para cifragem dos dados e a função *hash one-way SHA1* para a computação do MAC.

A Tabela 7.1 apresenta os tempos médios[24], em milisegundos, para inicialização do Tahiti e registro no *MAFFinder*, para executar o agente móvel *two-hop* bumerangue. Esta medida permite avaliar o custo de processamento adicional relativo ao tempo gasto nas interações sobre a rede para o estabelecimento do contexto SSL. É interessante observar que o tempo para conexão com o *MAFFinder* e para a primeira execução do agente (primeiro envio e recebimento), quando o SSL está sendo utilizado, é relativamente alto. O tempo para inicialização do Tahiti com SSL é aproximadamente duas vezes maior do que sem o SSL. Na primeira execução do agente usando o SSL, observa-se uma degradação de 24 % quando comparado com o caso sem SSL. Isto ocorre devido ao custo com o *handshake* SSL que usa criptografia assimétrica. No entanto, para a segunda execução do agente, o tempo com SSL de aproximadamente 113 ms é considerado satisfatório.

Cenário 1 - Rede Local	Tempo Médio (ms)
Inicialização do Tahiti e Registro no *MAFFinder* sem SSL	1016
Inicialização do Tahiti Registro no *MAFFinder* com SSL	2019
Primeira execução do agente *two-hop* bumerangue (sem SSL)	611
Primeira execução do agente *two-hop* bumerangue (com SSL)	800
Segunda execução do agente *two-hop* bumerangue (sem SSL)	38
Segunda execução do agente *two-hop* bumerangue (com SSL)	113

Tabela 7.1: Comparação da Latência com o uso do Protocolo SSL

Outro experimento realizado foi a medição do tempo de envio e recebimento de um agente (Figura 7.9.1), carregando um vetor de *bytes* de diferentes tamanhos, entre duas plataformas de agentes. O tamanho dos vetores de *byte* variaram de 256 *bytes* a 1 *Mbytes*. Pode-se observar que os tempos medidos para a segunda execução do agente *two-hop* bumerangue, usando o SSL, sofrem uma degradação média de 227% para vetores de até 4Kbytes, de 136% para vetores entre 8 *Kbytes* e 128 *Kbytes* e de 37% para vetores entre 256 *Kbytes* e 1 *Mbytes*. Estes resultados refletem o custo do suporte de segurança para cifragem e decifragem de mensagens transmitidas sobre a rede. Quando os agentes carregam vetores maiores, a degradação com o uso do SSL não é tão expressiva já que o custo para serialização e desserialização do agente e latência da rede também é elevado.

[24]Este experimento foi repetido dez vezes e a média aritmética dos tempo tempo foi calculada.

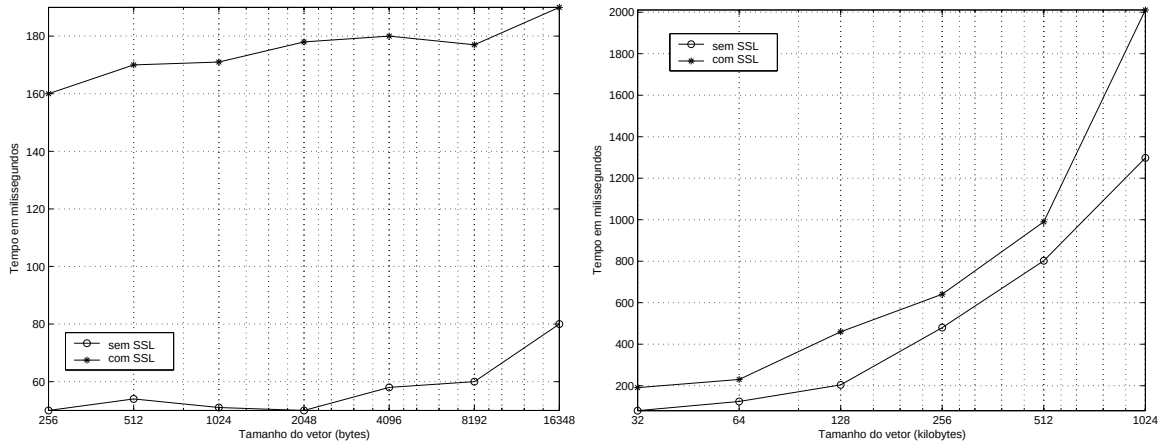


Figura 7.10: Gráficos Comparativos dos Tempos de Envio de um Agente Carregando um Vetor de Tamanho Variável com e sem SSL

7.9.2 Cenário 2: Plataforma Aglets + Protocolo de Autenticação Mútua + Protocolo SSL

Como para este cenário, o tamanho do agente não influencia no custo adicional de processamento imposto pelo protocolo de autenticação mútua, somente medidas de latência foram realizadas. Os algoritmos utilizados nos testes do protocolo de autenticação mútua foram o *SHA-1* e o *RSA- chave de 1024* (assinatura digital). A Tabela 7.2 apresenta os tempos médios para a primeira execução do agente *two-hop* bumerangue, considerando o uso do protocolo de autenticação mútua das plataformas de agentes e do protocolo SSL. Nota-se que o custo com o uso do protocolo de autenticação mútua é, aproximadamente, três vezes maior do que quando não ocorre a autenticação. Quando o protocolo SSL é adicionado, esta relação sobe para três vezes e meio. Esta expressiva degradação é devido às várias trocas que ocorrem no protocolo de autenticação mútua e que envolvem operações de assinatura digital. Vale ressaltar que este custo elevado acontece somente quando as plataformas fonte e destino não estabeleceram uma confiança mútua, ou seja, para as próximas interações subseqüentes entre as plataformas, somente a degradação do desempenho com o uso do SSL é que ocorrerá.

Cenário 2: Rede Local	Tempo Médio (ms)
Envio do agente two-hop bumerangue	704
Envio do agente two-hop bumerangue sem SSL com Autent. Mútua	1771
Envio do agente two-hop bumerangue com SSL com Autent. Mútua	2173

Tabela 7.2: Comparação da Latência com o uso do Protocolo de Autenticação Mútua de Plataformas

7.9.3 Cenário 3: Autenticação de Agentes Móveis que Utilizam Repositórios Seguros de Dados

O objetivo dos experimentos realizados neste cenário de testes foi o de medir o custo adicional para o envio e recebimento de um agente móvel *two-hop* bumerangue, que usa os

repositórios seguros de dados mais o custo do mecanismo de autenticação de agentes que incluem a verificação desses repositórios. Para este cenário, os seguintes algoritmos foram utilizados: *SHA1* e *RSA (1024)* para assinatura digital, *RSA (1024)* para distribuição de chave e *3DES* para cifragem de dados.

No primeiro experimento, um agente móvel foi implementado usando o `RORepository` da biblioteca *AgentSec*. Quando o agente é iniciado, o repositório somente-leitura é criado, a *String* "Teste"é adicionada no repositório, e então o agente migra para a plataforma-destino. Antes de ser executado na plataforma-destino, o agente é autenticado (a integridade do repositório é verificada). O agente então retorna para a plataforma de origem para que novamente a integridade do repositório seja verificada. A Tabela 7.3 apresenta a média dos tempos para um agente que carrega a *String* "Teste"sem proteção e para um agente que carrega a *String* protegida dentro do `RORepository`. Ainda considerando este mesmo agente, só que agora carregando um vetor de *bytes* de tamanho variável (256 *bytes* a 1 *Mbyte*), outras medidas foram realizadas para avaliar os tempos de envio e recebimento deste agente (ver Figura 7.11).

Envio e Recebimento do Agente (com autenticação)	Tempo (ms)
Sem `RORepository`	104.3
Com `RORepository`	231.1
Com `RORepository` e `DDRepository`	421.9

Tabela 7.3: Comparação da Latência quando o agente usa o *RORepository* e o *DDRepository*

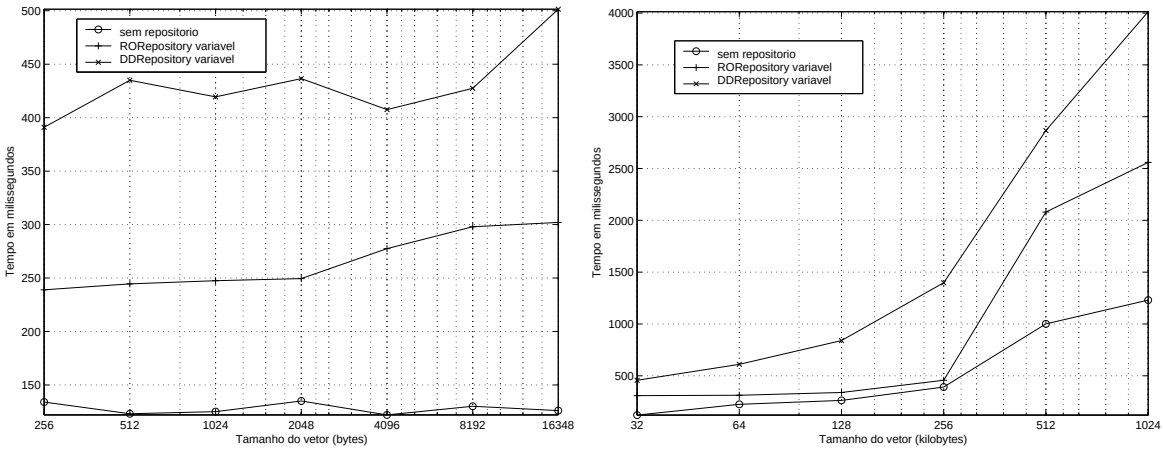


Figura 7.11: Gráficos Comparativos de um Agente com e sem os repositórios *RepositoryRO* e *RepositoryDD* de Tamanhos Variáveis

Para avaliar o custo introduzido com o uso do repositório de dados direcionados, que usa cifragem simétrica dos dados e cifragem assimétrica de chaves temporárias, um agente que carrega o `RORepository` e o `DDRepository`[25], com apenas uma entrada de dados, foi implementado. A Tabela 7.3 apresenta os resultados da latência do envio e recebimento

[25]Sempre que um `DDRepository` ou um `PRRepository` for anexado a um agente, o `RORepository` precisa ser também anexado, pois este é quem carrega os objetos `Credentials` e `QoP` necessários para a verifi cação destes repositórios

do agente móvel quando a entrada de dados é a *String* "Teste"e a Figura 7.11 apresenta os resultados quando o dado é um vetor de *bytes* com tamanho variável. Lembrando que os resultados apresentados na Tabela 7.3 e na Figura 7.11 inclui o processo de autenticação do agente (verificação dos repositórios nas duas plataformas). Observa-se nestes experimentos uma expressiva degradação do desempenho. Isso se deve à cifragem dos dados que mesmo sendo simétrica provoca um custo elevado ao desempenho.

Para realizar os experimentos com o `PRRepository`, um agente foi implementado e uma configuração mais complexa das plataformas de agentes se fez necessária. A Figura 7.12 ilustra o esquema de saltos do agente `testePR`, em que seis plataformas são visitadas pelo agente (três em cada máquina). O agente `testePR` partiu de sua plataforma de origem com um `RORepository` e foi migrando de uma máquina para outra, alternadamente, adicionando no `PRRepository` os resultados parciais introduzidos por cada plataforma visitada. Conforme ilustrados na figura, tempos parciais foram obtidos a cada dois saltos até obter o tempo total quando o agente retorna para a plataforma de origem. Em cada plataforma visitada, o agente é autenticado. A Tabela 7.4 compara os tempos médios do percurso do agente `testePR`, quando o resultado parcial adicionado por cada plataforma foi a *String* "Teste", considerando os três protocolos propostos para inserção de resultados parciais.

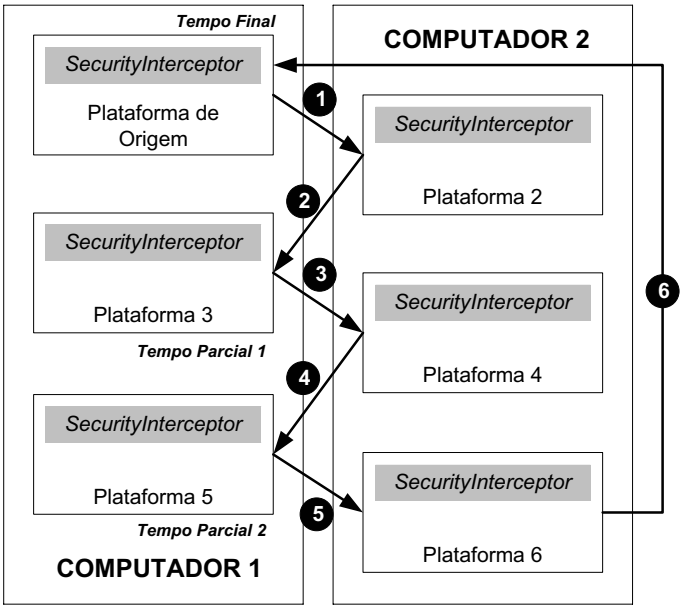


Figura 7.12: Esquema de Saltos do Agente *TestePR*

Agente `TestePR`	Tempo Parcial 1 (saltos 1 e 2)	Tempo Parcial 2 (saltos 3 e 4)	Tempo Parcial 3 (saltos 5 e 6)	Total (ms) (saltos de 1 a 6)
Sem `PRRepository`	120.8	112.3	107.3	340.4
Protocolo A	861	925.4	2937.5	4723.9
Protocolo B	775.5	436.9	510.3	1722.7
Protocolo C	803.1	824.8	4072.6	5700.5

Tabela 7.4: Comparação da Latência quando o agente usa o *RORepository* e o *PRRepository*

Analisando os tempos apresentados na Tabela 7.4, observa-se o custo introduzido pelos processos de cifragem, assinatura digital e cálculo da relação de encadeamento (*hash*), de acordo com a configuração definida em cada protocolo do MASS. No caso do protocolo A, em que os resultados parciais inseridos são cifrados e depois assinados, além do cálculo da relação de encadeamento, observa-se um tempo elevado para o processamento dessas operações nas duas primeiras parciais. Vale lembrar que este protocolo tem a característica de permitir que os resultados parciais introduzidos anteriormente possam ser verificados (autenticados). Este tempo aumentou na última parcial (retorno à plataforma de origem) já que neste ponto cinco resultados parciais foram verificados no processo de autenticação e ocorreu ainda a decifragem dos resultados parciais.

Quando examinamos os resultados com o protocolo B, constatamos que os tempos mais baixos, quando comparados com os resultados do protocolo A, se devem ao fato de não haver cifragem dos resultados parciais[26]. Quando comparados com os do protocolo A, os tempos obtidos com o uso do protocolo C, nas duas primeiras parciais, são discretamente menores. Isto se deve ao fato de que este protocolo, apesar de aplicar cifragem e assinatura digital, não permite que a integridade dos resultados seja verificada em plataformas intermediárias. Como a autenticação dos resultados parciais inseridos é verificada somente na plataforma de origem, o tempo da última parcial é bem elevado.

Semelhante ao experimento anterior, neste último experimento, o agente teste percorre as plataformas especificadas (ver Figura 7.12) recolhendo resultados parciais que variam de 256 *bytes* a 1 *Mbyte*. Ou seja, quando cada resultado parcial recolhido for de 256 *bytes*, ao retornar para a sua plataforma de origem, o `PRRepository` terá cinco entradas de 256 *bytes* (1280 *bytes*). Durante a execução do experimento acima, conclui-se que um agente carregando um `PRRepository`, usando o protocolo A ou C, não consegue terminar sua viagem se o tamanho dos dados carregados exceder a um certo limite (ver Figura 7.9.3. Isto ocorreu, pois o processo de verificação da integridade do repositório necessitava de mais memória virtual do que dispunham os computadores utilizados nos testes[27]. No caso do protocolo A, o limite encontrado ocorreu quando o resultado parcial recolhido (vetor) tinha 1 *Mbyte* e quando o agente já se encontrava na plataforma de origem para a última verificação do `PRRepository` (com 5 *Mbytes*). Já no caso do protocolo C, a situação limite encontrada ocorreu quando o agente havia recolhido um vetor de 512 *Kbytes* em cada plataforma e se encontrava na plataforma de origem tentando realizar a última verificação do repositório (com *2.5 Mbytes*). Vale ressaltar ainda que quando o resultado parcial foi de 1 *Mbyte*, o agente só foi interrompido quando este já tinha retornado a sua plataforma de origem, ou seja, quando `PRRepository`, tinha 5 *Mbytes* de resultados protegidos (ver Figura 7.9.3).

Diante do exposto acima, conclui-se que o uso do processo de cifragem dos resultados parciais encapsulados (protocolos A e C) acarreta um custo computacional bastante elevado,

[26] Mesmo havendo ainda a verifi cação da integridade dos resultados parciais em cada plataforma visitada.

[27] Uma exceção do tipo `java.lang.OutOfMemoryError` foi gerada durante tal processo e o agente não pôde completar sua viagem.

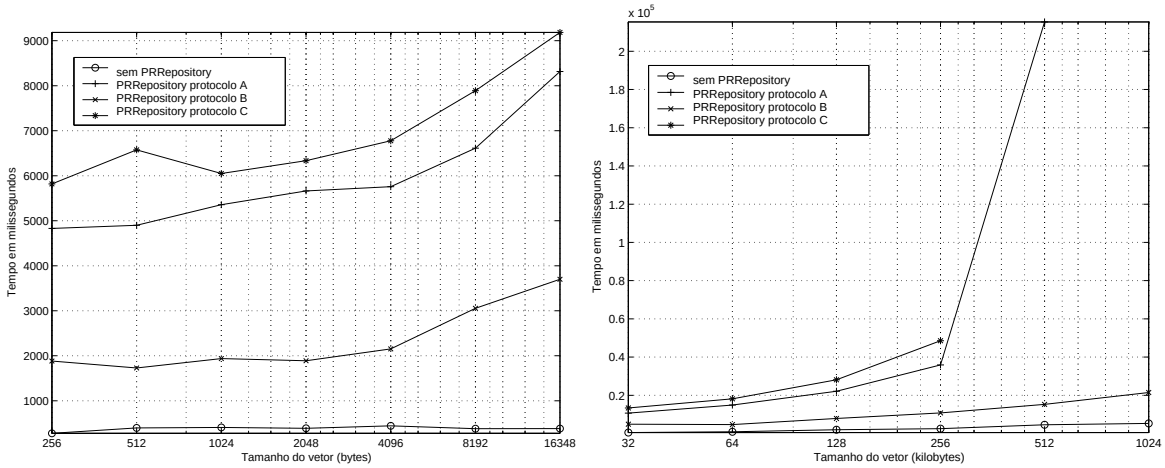


Figura 7.13: Gráficos Comparativos de um Agente com e sem o *RepositoryPR* de Tamanhos Variáveis

principalmente, quando o resultado encapsulado ultrapassa 512 Kbytes e quando mais de quatro plataformas inserem resultados neste repositório. Por este motivo, é recomendável que para resultados parciais grandes (acima de 512 Kbytes) o protocolo B seja adotado para inserção de resultados no *PRRepository*, ou que somente quatro resultados sejam inseridos quando os protocolos A e C forem os utilizados.

7.10 Diretrizes Gerais para Seleção dos Mecanismos de Segurança do Protótipo

Com a coleção de mecanismos de segurança propostos no *MASS*, os usuários da tecnologia de agentes móveis estão livres para desenvolver aplicações distribuídas que podem se beneficiar das vantagens dessa tecnologia. Entretanto, o uso dessa coleção de mecanismos de segurança introduz restrições de desempenho que podem ditar decisões de projeto e negar os benefícios do uso de agentes móveis para certas aplicações.

A seleção dos mecanismos de segurança, a ser empregada em uma aplicação baseada em agentes móveis, deve ser cuidadosamente considerada durante a fase de projeto do ciclo de desenvolvimento desta aplicação e não deve ser adicionada somente no final do ciclo. Questões de segurança podem determinar quais agentes serão móveis e quais permanecerão estáticos em uma aplicação, bem como determinar as funções que um agente móvel deve executar e as que este nunca poderá exercer. Uma análise dos requisitos de segurança da aplicação e uma análise de custo-benefício (uso de mecanismos de segurança) devem ser cuidadosamente realizadas para assim definir quais os mecanismos de segurança do *MASS* serão selecionados para uma dada aplicação distribuída.

Visando auxiliar os desenvolvedores de aplicações, propõem-se nesta seção algumas diretrizes para a seleção dos mecanismos de segurança do protótipo com base nos requisitos

Diretriz	Requisitos Desejáveis de segurança	Mecanismo Recomendável
1	Garantir que, quando o agente estiver trafegando pela rede, o seu código e estado estarão protegidos	Protocolo SSL integrado à plataforma Aglets
2	Estabelecer a confi ança mútua entre plataformas que enviam e recebem agentes móveis	Protocolo de Autenticação Mútua
3	Proteger a integridade do código do agente quando este visita uma plataforma maliciosa	Assinatura Digital do Código do Agente
4	Proteger os dados somente-leitura carregados pelo agente de plataformas maliciosas	Agente carregando os dados no RORepository
5	Proteger a confi dencialidade e a integridade de dados direcionados para plataformas específi cas	Agente carregando os dados no DDRepository
6	Criar registros das plataformas visitadas pelo agente, minimizando ataques de repudiação	Agente carregando um PathRegsiter
7	Garantir a integridade dos resultados parciais encapsulados pelo agente. A integridade dos dados deve ser verifi cada somente na plataforma home	Agente encapsulando os dados em um PRepository - Protocolo B - plataforma de origem
8	Garantir a integridade dos resultados parciais encapsulados pelo agente. A integridade dos dados deve ser verifi cada por qualquer plataforma	Agente encapsulando os dados em um PRepository - Protocolo B - verif. publicamente
9	Garantir a integridade e a confi dencialidade dos dados encapsulados pelo agente. A integridade dos dados deve ser verifi cada somente na plataforma de origem	Agente encapsulando os dados em um PRRepository protocolo A - plataforma de origem
10	Garantir a integridade e a confi dencialidade dos dados encapsulados pelo agente. Sendo que a integridade dos dados é verifi cada em cada plataforma visitada	Agente encapsulando os dados em um PRRepository protocolo A - verif. publicamente
11	Garantir a integridade e a confi dencialidade dos dados encapsulados pelo agente. Garantir ainda a privacidade das plataformas que inserirem resultados. A integridade dos dados dever ser verifi cada na plataforma de origem	Agente encapsulando os dados em um PRRepository Protocolo C

Tabela 7.5: Diretrizes para Seleção dos Mecanismos de Segurança do *MASS*

de segurança da aplicação e nos resultados da análise de desempenho dos mecanismos. As diretrizes são apresentadas na Tabela 7.5.

Para a escolha de qual protocolo será usado para inserção de resultados parciais, vale lembrar que o custo computacional com o uso desses repositórios é elevado, por isso é muito importante contrapor os requisitos de segurança com o custo do desempenho. O custo computacional dos mecanismos recomendados das diretrizes que tratam do `PRRepository` (diretrizes de 7 a 11) é maior na ordem crescente das diretrizes. Ou seja, quando os requisitos da diretriz *10* for o desejado pela aplicação, deve-se lembrar que para este pior caso, o *PRRepository* não se mostra adequado para agentes cujo itinerário inclui muitas plataformas a serem visitadas (mais de 10 plataformas) ou que recolha dados com tamanho superiores a 512 *KBytes* e que percorra mais de quatro plataformas.

7.11 Integração do Protótipo a uma Aplicação Distribuída

Dentre os domínios de aplicações para os quais os agentes móveis podem trazer grandes benefícios e que já apresentam resultados significativos, podem ser citados: o da consulta de informação distribuída (Knudsen, 1995), o dos serviços de telecomunicações (Breugst e Magedanz, 1998a; Magedanz *et al.*, 1996), o das redes ativas (Breugst e Magedanz, 1998b) e o do comércio eletrônico (Dasgupta *et al.*, 1998). Diversas aplicações de comércio eletrônico baseadas em agentes têm sido propostas e estão sendo desenvolvidas por diferentes áreas de negócios, incluindo negociações de contrato, comercialização de serviços, leilões e cooperações de organizações (CNOs- *Collaborative Networked Organizations*).

O protótipo do *MASS* foi utilizado na aplicação de Busca e Seleção de Parceiros para Formação de Empresas Virtuais — *MobiC-II system* (Schmidt, 2003), como parte da contribuição do Departamento de Automação e Sistemas ao projeto IFM (Instituto Fábrica do Milênio). Já na fase de projeto desta aplicação baseada em agentes móveis, requisitos de segurança foram considerados e influenciaram na arquitetura híbrida de agentes adotada (agentes móveis e estacionários).

7.11.1 *MobiC-II*: Sistema de Busca e Seleção de Parceiros para Formação de Empresas Virtuais

Com o desafio de ampliar sua participação no mercado de negócios sem alterar drasticamente suas estruturas, muitas organizações estão adotando o modelo de empresas virtuais. Uma empresa virtual corresponde a uma união temporária de empresas, participantes de uma organização virtual (OV), que se agregam visando atender os requisitos de uma dada oportunidade de negócios (Camarinha-Matos e Afsarmanesh, 1999). Um exemplo de organização virtual é ilustrado na Figura 7.14, em que a mesma é composta por nove empresas reais. As empresas que recebem dos clientes as oportunidades de negócio (ON) atuam como um *broker* independente centralizando a interação com o cliente (A ou B) e são as responsáveis por todo o processo de busca e seleção de parceiros (Rabelo *et al.*, 2003). Isto permite que diversas oportunidades de negócios possam ser tratadas simultaneamente dentro de uma organização virtual. Diante de uma oportunidade de negócio apresentada por um cliente (cliente B), tem-se um cenário com a possibilidade de constituição de duas empresas virtuais (EV2 e EV3). Depois de um processo de avaliação conduzido pelo *broker*, um conjunto de empresas é selecionado e então a empresa virtual é criada A cooperação entre estas empresas é garantida pela automação e informatização de grande parte de suas plantas industriais, deixando-as acessíveis via Internet.

Visando alcançar a competitividade almejada por uma organização virtual, duas questões precisam ser consideradas durante a fase de criação das empresas virtuais: a agilidade na apresentação das oportunidades de negócio ao grupo de empresas da organização virtual, e

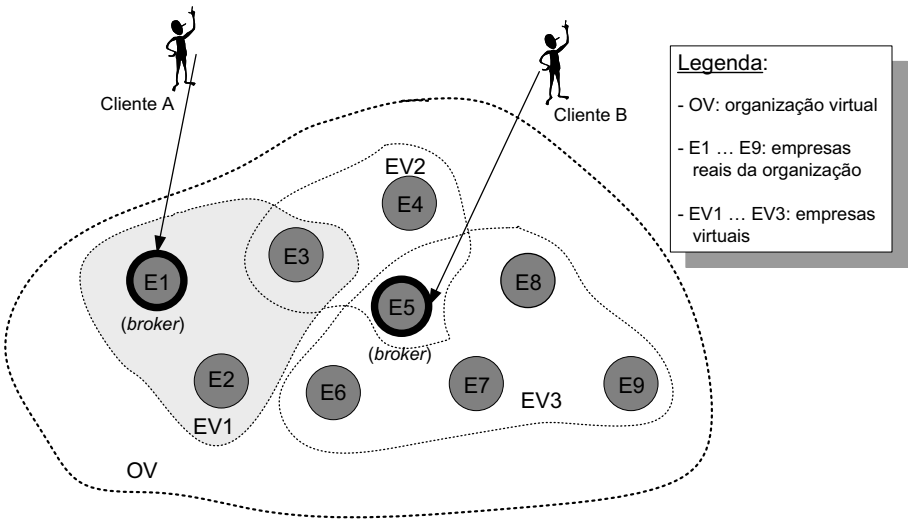


Figura 7.14: Cenário para a Seleção de Empresas Virtuais

a eficiência na formação e análise das possíveis empresas virtuais a serem constituídas para atender a essas oportunidades. Além disso, devido às características de dinamismo e de heterogeneidade das infraestruturas das empresas integrantes, são necessários mecanismos que tratem desses aspectos. Schmidt (2003) explora a complexidade da formação de empresas virtuais, apresentando uma abordagem para auxiliar a fase de criação de empresas virtuais baseada em uma arquitetura híbrida de agentes. Esta abordagem explora os benefícios do paradigma de agentes móveis para garantir a eficiência desejada. Entre os benefícios, destacam-se: a redução da carga da rede com a possibilidade de interação local entre as empresas e o fato de poder operar de forma assíncrona e autônoma, possuindo uma independência das características dos canais de comunicação. Porém, para beneficiar-se dessas características, as situações em que os agentes móveis foram empregados foram cuidadosamente avaliadas e um esquema de segurança — *MASS*— precisou ser introduzido para fortalecer o processo de construção de confiança na formação das EVs. Os agentes estacionários, que representam cada empresa real, são os responsáveis pela interação com os sistemas legados das empresas para a obtenção das informações desejadas (Schmidt, 2003).

Um protótipo que implementa o processo de busca e seleção de parceiros— *MobiC-II* (Schmidt, 2003) — foi desenvolvido para um cenário de testes composto pela organização virtual *Techmoldes*, formada por nove pequenas e médias empresas de moldes de Caxias do Sul, RS, cujos membros cooperam para aprimorar sua competividade global. Quando estão cooperando dentro da *Techmoldes*, cada membro permanece independente e autônomo, podendo, inclusive, fazer negócios fora da OV.

No *MobiC-II*, os agentes móveis têm como função deslocar-se pelas empresas em busca de informações para a formação de EVs ou para negociar melhores preços ou menores prazos de entrega (ver Figura 7.15). Estes são os responsáveis pela ligação entre o *broker*, que representa a organização virtual e que receberá as oportunidades de negócios, e os agentes estacionários que representam cada empresa real da organização virtual.

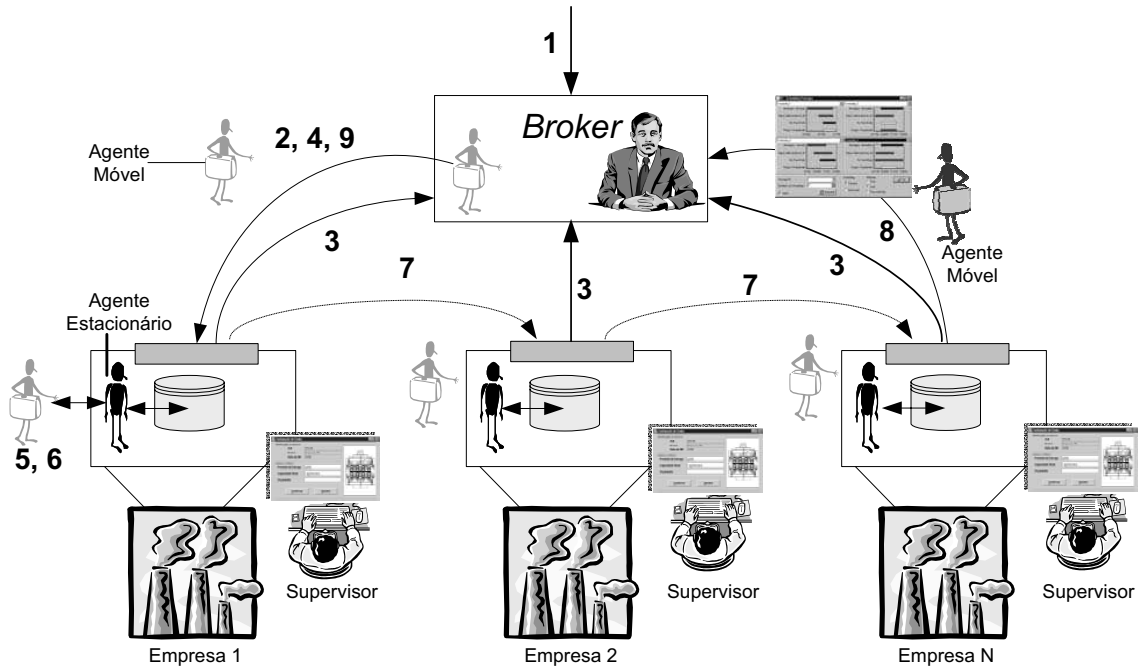


Figura 7.15: Sistema *MobiC-II*

Conforme descrito em Rabelo *et al.* (2003) e ilustrado na Figura 7.15, um *broker*, ao receber uma oportunidade de negócio (ON), deve identificar as empresas em potencial que podem fornecer cada molde (**passo 1** da Figura 7.15). Um resumo da especificação do(s) molde(s) é imediatamente enviado para as empresas identificadas (**passo 2**). Cada empresa que recebe o anúncio resumido da ON avalia sua capacidade e seus interesses preliminares e envia uma resposta para o *broker*, que pode ser *sim* (expressando seu interesse) ou *não* (**passo 3**). O *broker* recebe as respostas e envia um agente móvel para as empresas interessadas, fornecendo a especificação completa da ON e a lista das empresas a visitar (**passo 4**). Quando o agente móvel chega na primeira plataforma, este interage com o agente estacionário da empresa perguntando por sua capacidade de produção, disponibilidade de recursos e tempo de entrega (**passo 5**). O agente local, atuando como representante da empresa, obtém estas informações de seus sistemas legados e das base de dados locais. Após isto, o agente móvel solicita ao supervisor local informações sobre o preço para produção do molde já que esta é uma informação crítica no setor de moldes. Um processo de negociação local pode ser executado (**passo 6**). Então, o agente móvel migra para a próxima empresa da lista carregando estas informações (**passo 7**). Este processo é repetido até que todas as empresas candidatas na lista sejam visitadas. Depois de obter as informações necessárias, o agente móvel deve retornar para o *broker* (**passo 8**). Após receber as propostas das empresas, o agente *broker* compõe o conjunto possível de empresas virtuais, devidamente escalonadas, a partir de indicadores de desempenho previamente definidos para avaliá-las, e um humano (*broker*) elege a combinação mais adequada. O critério adotado no cenário *Techmoldes* constituiu-se o menor preço global e o menor tempo de entrega. Por fim, o *broker* envia uma mensagem indicando a composição vencedora para as empresas (**passo 9**).

Três classes de agentes compõem o sistema *MobiC-II* (Schmidt, 2003):

- **Agente *Broker***: é um agente estacionário responsável por receber uma oportunidade de negócio, distribuí-la para empresas em potencial, enviar agentes móveis para estas empresas e montar as possíveis combinações de empresas.
- **Agente Móvel**: é um agente responsável por entregar uma oportunidade de negócio às empresas da OV, negociar localmente com estas e, após percorrer todas as empresas, retornar ao *broker* com as informações coletadas. Este agente pode assumir diferentes papéis (missões): atuar como um simples agente *mensageiro* de informações, como um *pesquisador* de dados ou então atuar como um *negociador*, sendo capaz de tomar decisões e realizar a negociação sem depender de ordens enviadas pelo agente *Broker*, no decorrer da sua tarefa. Assim, papéis são criados para o agente, conforme atuação pretendida.
- **Agente Empresa**: é um agente estacionário que representa uma empresa e é responsável por receber uma ON, avaliá-la, acessar sua base local de dados para obter as informações requeridas e responder à ON para o agente móvel.

Duas plataformas diferentes de agentes foram usadas na implementação do *MobiC-II*. Os agentes móveis foram codificados com a linguagem Java e a plataforma Aglets da IBM (IBM, 1996) foi utilizada e os agentes estacionários foram codificados em C++ e a plataforma MASSYVE-KIT[28] foi adotada. Para garantir a interoperabilidade entre estes agentes, o CORBA foi utilizado como *middleware* de comunicação.

O processo de construção de confiança é apontado como um dos problemas mais difíceis para ser contornado pelos desenvolvedores de soluções para empresas virtuais (Rabelo *et al.*, 2003). O sistema *MobiC-II* considera a necessidade de se ter mais de um Broker atuando dentro de uma OV. E isso traz vantagens para o sistema pelo fato de reduzir a concentração de atividades em um único elemento e de possibilitar uma hierarquia do tipo descentralizada, auxiliando na construção da confiança entre as empresas participantes. Entretanto, mesmo que os membros da OV se conheçam e sejam conscientes de que todos são candidatos para uma oportunidade de negócio, ainda há uma relutância em compartilhar alguns tipos de informações como: preço, tempo de entrega e capacidades de produção. Alguns problemas culturais, éticos e administrativos relacionados a infra-estrutura de tecnologia, têm sido apontados como obstáculos para ampla adoção do paradigma de empresas virtuais pelas organizações, segundo Camarinha-Matos e Afsarmanesh (2002). Diante disso, mecanismos de segurança que garantam a confidencialidade, integridade e disponibilidade das informações, conforme especificado na política de segurança da organização virtual, devem ser introduzidos para a construção da confiança. A seção, a seguir, apresenta como a construção de

[28] http://www.gsigma-grucon.ufsc.br/massyve/mkit.htm

confiança, em uma arquitetura baseada em agentes móveis, pode ser enriquecida pelo uso de mecanismos de segurança no processo de busca e seleção de parceiros para formação de empresas virtuais.

7.11.2 Integração do *MASS* ao *MobiC-II*

Identificação das Ameaças de Segurança

Nos passos do sistema *MobiC-II* que utilizam agentes móveis — passos 2, 4, 5, 6, 7 e 8 (ver Figura 7.15), as ameças de segurança contra os agentes — mascaramento, negação de serviço, intromissão e modificação não autorizada — e contra as plataformas de agentes móveis — mascaramento, negação de serviço, repudiação e acesso não autorizado, são também encontradas no Cenário TechMoldes. As ameças contra o canal de comunicação que podem comprometer tanto o envio de agentes quanto o envio de mensagens entre as plataformas, tais como, modificação não autorizada, intromissão, mascaramento e repudiação, estão presentes nos passos 1, 2,3, 4, 7, 8 e 9 do sistema *MobiC-II*.

Política de Segurança para o Cenário TechMoldes

Uma política de segurança organizacional é um conjunto de regras e práticas impostas por uma organização que estabelece os limites de operação dos usuários do sistema para proteger os dados sensíveis da organização. Na fase de projeto do sistema *MobiC-II*, definiu-se uma política de segurança para o cenário *TechMoldes* e uma resposta planejada às ameaças do sistema — os objetivos de segurança do *MASS* para o *MobiC-II*. Um resumo das regras da política de segurança definida são apresentadas, a seguir:

- P1: a integridade dos dados somente-leitura carregados pelos agentes móveis (ex.: especificação resumida e especificação completa da ON) deve ser garantida pelo *MASS*;
- P2: somente as empresas da OV *TechMoldes* devem ter acesso ao resumo e à especificação completa da ON;
- P3: somente as empresas da OV *TechMoldes* podem assumir o papel de *broker* e com isso somente estas poderão enviar os agentes móveis com as especificações das ONs e os agentes pesquisadores e negociadores;
- P4: o *MASS* deve controlar o acesso dos agentes móveis a informações sensíveis das plataformas;
- P5: a autenticidade da origem de um agente móvel (quem o criou) deve ser verificável;
- P6: somente as empresas participantes podem responder a uma dada oportunidade de negócio (através dos agentes móveis negociador ou pesquisador). Essas propostas coletadas devem ser reveladas somente para o *broker* e a sua integridade deve ser garantida;

- P7: a integridade e a autenticidade da origem de todas as mensagens trocadas entre as plataformas de agentes móveis devem ser garantidas;
- P8: a integridade e a confidencialidade dos agentes móveis, quando estes estão sendo enviados pelo canal de comunicação, devem ser garantidas pelo *MASS*;
- P9: o código de um agente móvel só pode ser modificado pelo seu proprietário;
- P10: uma empresa não pode negar que recebeu uma dada oportunidade de negócio;
- P11: uma empresa não pode negar que apresentou uma proposta em resposta a uma dada oportunidade de negócio;
- P12: todos os parceiros da EV devem ter acesso a mensagem final de criação da EV.

Análise dos Mecanismos de Segurança para o *MobiC-II*

Após identificar as ameaças e definir uma política de segurança organizacional, foram analisados os mecanismos de segurança suportados no *MASS* e identificados os mecanismos necessários para minimizar ou eliminar a exploração de uma ou mais vulnerabilidades que resultariam no comprometimento da construção da confiança no sistema *MobiC-II*. Estes mecanismos, com a indicação das regras que estes satisfazem na política de segurança, estão listados na Tabela 7.6.

Mecanismos de Segurança	Regras
Uso do repositório de dados somente-leitura (`RORepository`) para proteger os anúncios resumido e completo	P1
Uso do repositório de resultados parciais (`PRRepository` - protocolo A) para proteger as propostas apresentadas pelas empresas candidatas a formar a EV	P6 e P11
Uso do repositório de dados direcionados (`DDRepository`) para proteger o agente mensageiro enviado pelo *broker* para os parceiros da EV	P12
Uso do objeto `PathRegister` para registrar o itinerário de viagens dos agentes (mensageiro, pesquisador e negociador)	P2 e P10
Autenticação de agentes móveis — verificação da integridade dos repositórios e `RORepository` e `PRRepository`	P1 e P6
Autenticação de agentes móveis —- verificação da assinatura do agente móvel	P3, P5 e P9
Procedimentos para Geração de Domínios de Proteção	P4
Estabelecimento de canal seguro — autenticação mútua	P3
Estabelecimento de canal seguro — uso do protocolo SSL	P7 e P8

Tabela 7.6: Mecanismos de Segurança para o Sistema *MobiC-II*

Após analisar a funcionalidade do agente *negociador* e considerando aspectos de desempenho, concluiu-se que o uso do protocolo A, que garante a integridade e a confidencialidade das propostas encapsuladas pelo agente e que permite que a integridade seja verificada por todas as empresas da OV, é o mais adequado para o cenário *TechMoldes*. Como as propostas recolhidas pelo agente são documentos XML com tamanho variando de 1 *Kbyte* a 4 *Kbytes*

e no máximo oito plataformas poderão apresentar propostas, o custo computacional como o uso do `PRRepository` com o protocolo A não nega os benefícios com o uso do agente móvel negociador.

7.12 Considerações Finais

A avaliação de um sistema distribuído com características abertas, que utiliza componentes COTS (*Commercial Off-The-Shelf*) e que abrange a grande rede mundial (Internet) é uma tarefa árdua. Em geral, em razão da complexidade desses sistemas e das limitações das técnicas de verificação e validação, a **segurança** nesses sistemas é muito difícil de ser provada ou estimada. Em muitos mecanismos de segurança, a fundamentação está na dificuldade computacional que, ao contrário do que ocorre em sistemas confiáveis e tolerantes a falhas, não pode nem mesmo estabelecer uma previsibilidade probabilística de comportamento. Ou seja, as falhas dos mecanismos de segurança se devem à capacidade computacional dos atacantes e à dificuldade de prever o comportamento humano. Na prática, não é possível construir sistemas que sejam garantidos como sendo seguros ou que permaneçam seguros todo o tempo.

Espera-se que as políticas de seguranças sejam internamente consistentes e que reflitam os requisitos das organizações onde estas serão aplicadas. Assim como espera-se que os mecanismos de segurança funcionem corretamente e executem as funções para as quais eles foram planejados. Esses aspectos críticos da confiança nas políticas e nos mecanismos são difíceis de serem quantificados ou analisados (Bishop, 2003). A noção de quanto um sistema é seguro está baseada em julgamentos e não em medidas exatas de segurança, ou seja, qualquer parecer sobre a segurança de um sistema é muito mais uma avaliação qualitativa do que quantitativa.

Entretanto, há esforços no sentido de avaliar o nível de segurança de sistemas ou produtos. A avaliação é um processo em que as evidências para garantias de segurança são reunidas e analisadas contra alguns critérios para funcionalidade e garantia da segurança. Esta pode resultar em uma medida de confiança que indica quão bem um um sistema atende alguns critérios particulares. Os critérios usados depende dos objetivos da avaliação e da tecnologia de avaliação usada. A metodologia de avaliação deve refletir o grau de coerência entre o modelo ideal de funcionalidade e garantias de segurança do sistema (política de segurança) e o real (o implementado).

Duas metodologias para avaliação de segurança vem sendo comumente utilizadas: os Critérios Comuns (CC)(ISO/IEC, 1999) e o Gerenciamento de Risco (Stoneburner *et al.*, 2001), proposto pelo NIST (National Institute of Standards and Technology). Os Critérios Comuns (*Commnon Criteria* - CC) para avaliação qualitativa da segurança, padrão ISO 15408, são o resultado do esforço de várias organizações internacionais para desenvolver critério único para avaliação da segurança e especificação de sistemas e produtos de tecnologias de informação. Este padrão deriva de padrões anteriores como o TCSEC, ITSEC (critério europeu),

CTCPEC (critério canadense), entre outros. A abordagem do CC define metodologias padronizadas para uma agência credenciada gerar uma certificação classificando o nível de segurança do sistema. A segunda abordagem é fundamentada na gerência de riscos e, ao contrário do CC, é baseada em uma sistemática, executada por um especialista, para atingir um nível aceitável de segurança do sistema. Através do gerenciamento de riscos, o especialista pode avaliar o nível de suscetibilidade a adversidades (ameaças e vulnerabilidades) do sistema[29].

Por mais que sejam validados por métodos formais e testes na fase de projeto, sistemas ou produtos de segurança, durante suas utilizações, geralmente apresentam falhas e vulnerabilidades. Ou seja, as políticas de seguranças devem assumir riscos quando consideram os possíveis danos de intrusões e os custos para prevenir tais intrusões nos sistemas. Portanto, abordagens de avaliação sempre concretas são aquelas baseadas na gerência de riscos.

O esquema de segurança proposto nesta tese, ao se preocupar em tratar dois problemas de segurança (proteção do agente e da plataforma de agentes), tornou-se bastante abrangente. Como conseqüência, o trabalho com a implementação dos mecanismos propostos e com a sua integração a uma aplicação real tornou-se bastante custoso e tomou proporções maiores do que as inicialmente previstas. A possibilidade da aplicação consistente de uma metodologia para avaliação da segurança do protótipo desenvolvido, tornou-se inviável diante do tempo disponível. Pois, para o emprego eficaz de uma dessas metodologias, um grande esforço e dedicação fazem-se necessários já que o protótipo construído está baseado em COTS (o que implicaria uma avaliação também de alguns destes componentes) e comprovação dos resultados destas avaliações (as evidências de garantias de segurança) exige uma extensa documentação.

7.13 Conclusões do Capítulo

Neste capítulo, a implementação do protótipo do *MASS* foi descrita visando demonstrar a flexibilidade e aplicabilidade do esquema de segurança para aplicações baseadas em agentes móveis. As tecnologias e ferramentas empregadas neste protótipo foram escolhidas considerando aspectos de escalabilidade, portabilidade, interoperabilidade, padronização, desempenho e forte aceitação na Internet. Dentre as tecnologias adotadas destacam-se o uso do SPKI, do protocolo SSL, da plataforma Java, de uma plataforma de agentes de código aberto (Aglets), do XML, de um mecanismo de controle de acesso baseado em papéis e o uso de criptosistemas assimétricos e simétricos.

Visando construir um esquema fácil de entender, usar e gerenciar, algumas interfaces gráficas foram construídas e diretrizes para a seleção dos mecanismos de segurança, de acordo

[29] A avaliação do risco envolve a identificação de vulnerabilidades, a avaliação do impacto do risco no sistema e as recomendações para redução do risco.

com os requisitos de segurança da aplicação, foram apresentadas. Pensando em aspectos de reusabilidade, a biblioteca para construção de agentes móveis protegidos foi desenvolvida para ser independente de plataformas de agentes podendo ser facilmente adaptada a diferentes plataformas de agentes. O protocolo de autenticação mútua de plataformas e o algoritmo de autenticação de agentes móveis, por estarem implementados em agentes, podem ser também reutilizados e adaptados a outras implementações de plataformas de agentes móveis.

Os resultados dos testes de desempenho apresentados neste capítulo comprovam a importância de se avaliar o custo computacional agregado ao uso de cada mecanismos com a funcionalidade da aplicação para que os benefícios com o uso de agentes móveis não sejam negados.

Conforme apresentado neste capítulo, a utilização dos mecanismos de segurança oferecidos pelo *MASS* contribuem para a construção da confiança no processo de busca e seleção de parceiros para formação de empresas virtuais (sistema *MobiC-II*) e também contornam quase todas as ameaças de segurança encontradas no cenário TechMoldes.

Capítulo 8

Conclusões

8.1 Revisão dos Objetivos

Um estudo geral sobre a segurança no paradigma de agentes móveis em sistemas abertos serviu de motivação para proposição de algumas soluções para tornar este modelo mais atrativo do ponto de vista da segurança. Diante disto, o objetivo geral deste trabalho foi desenvolver um esquema de segurança para aplicações baseadas em agentes móveis em sistemas abertos (*MASS*), a partir da identificação das ameaças a que tais aplicações estão sujeitas e das deficiências apresentadas pelas técnicas descritas na literatura para contorná-las.

A fim de que este esquema pudesse superar algumas dessas deficiências, o esquema de segurança procurou ser concebido de forma a: (1) adequar-se às dimensões dos sistemas e ao número de usuários e operações (escalabilidade); (2) adequar-se a ambientes heterogêneos (portabilidade e interoperabilidade); (3) combinar mecanismos complementares para conferir proteção mais efetiva às aplicações (compatibilidade); (4) possibilitar ser especializado através da seleção de um subconjunto de mecanismos mais adequados aos requisitos de segurança de cada aplicação (flexibilidade); (5) simplificar o seu uso e gerenciamento (simplicidade); (6) reduzir a degradação do desempenho provocada pelo uso dos mecanismos de segurança (desempenho).

A fim de que o esquema pudesse atender a estes requisitos, e de que as questões de pesquisa fossem respondidas, alguns objetivos específicos foram perseguidos. Após listar cada um deles, a forma como os mesmos foram atingidos será, sumariamente, descrita, a seguir:

- **Desenvolver um esquema de autenticação e de autorização, flexível e descentralizado, para proteção de plataformas de agentes móveis (*$MASS_{ap}$*) combinando mecanismos de prevenção e de detecção**.

  O esquema desenvolvido, considerando um modelo de agentes de múltiplos saltos (*multi-hop*) e de um ambiente heterogêneo e distribuído de plataformas, faz uso das

cadeias de confiança SPKI/SDSI e do conceito de federação SPKI para garantir um controle de autenticação e autorização totalmente descentralizado e flexível, possibilitando assim a implementação de políticas de segurança adequadas para sistemas abertos.

Antes do envio de agentes móveis, as plataformas envolvidas devem autenticar-se mutuamente para que, então, um canal seguro na infra-estrutura de comunicação possa ser criado. Com este procedimento, é possível prevenir que agentes sejam enviados e recebidos por plataformas não autenticadas e prevenir e/ou detectar ataques contra os agentes móveis, quando estes estão sendo enviados pelo canal de comunicação.

Quando um agente móvel é recebido em uma plataforma de agentes, o processo de autenticação deste agente é realizado por um autenticador *multi-hop* que estabelece o nível de confiança no agente (baseado na autenticidade do proprietário do agente, das plataformas que o agente visitou e, ainda, na lista definida das federações pelo proprietário do agente).

Durante a criação de um agente, a fim de expressar de forma flexível e escalável as suas credenciais, o proprietário fornece a este um conjunto de cadeias de autorização SPKI/SDSI, definindo assim os seus atributos de privilégios. A fim de prevenir que tais agentes tenham acesso e/ou modifiquem informações sigilosas contidas nas plataformas de agentes que os executam, foi desenvolvido um esquema para geração de domínios de proteção que, com base nas credenciais do agente (cadeias de autorização SPKI/SDSI), define os domínios de proteção e define quais as permissões que serão atribuídas a estes domínios.

- **Elaborar mecanismos de prevenção e de detecção para contornar e minimizar os riscos a que os agentes móveis estão suscetíveis quando estes estão se executando em plataformas maliciosas $MASS_{ma}$ e, desenvolver técnicas que permitam a identificação de plataformas maliciosas**.

  O esquema desenvolvido pretende prevenir e/ou detectar os ataques de plataformas maliciosas a partir da conjugação de diferentes mecanismos de segurança. Estes mecanismos são utilizados para verificar a reputação da plataforma de destino, para proteger a integridade do código do agente, para proteger a integridade e a confidencialidade do estado do agente (uso de repositório seguro de dados). De forma a fiscalizar comportamentos que devem ser impostos às plataformas de agentes, um mecanismo de controle social baseado em reputação foi proposto.

  O $MASS_{ma}$ auxilia o programador do agente a construir um agente móvel protegido (através do uso de interfaces gráficas) e ainda auxilia as plataformas visitadas a verificar, de forma transparente, a integridade do agente através do autenticador *multi-hop*. Durante o processo de autenticação do agente, quando uma violação for detectada, o mecanismo que aplica o controle social deve ser acionado para que a identificação de plataforma maliciosa seja possível.

- **Definir e implementar um protótipo que inclua o esquema de segurança desenvolvido, bem como as técnicas de fiscalização e detecção de plataformas maliciosas**.

  O protótipo do esquema (*MASS*) foi definido e implementado visando comprovar a sua flexibilidade e a viabilidade de sua utilização em aplicações distribuídas. A escolha das tecnologias e ferramentas que compõem o protótipo foi orientada por duas principais motivações: potencial para atender os requisitos definidos para o *MASS* e o seu grau de aceitação na Internet. Dentre as tecnologias, destacam-se: o SPKI/SDSI, a Plataforma Java, a Plataforma de Agentes Aglets, o XML e o Protocolo SSL. O mecanismo de controle social baseado em reputação e o mecanismo para verificação da reputação de plataformas, que dependem da infra-estrutura da teia de federações SPKI, não puderam ser implementados pela falta de um protótipo que suportasse totalmente a infra-estrutura necessária.

  De forma a avaliar o custo computacional dos mecanismos do protótipo do *MASS*, uma análise do desempenho foi realizada e, diante dos resultados obtidos, diretrizes para a seleção dos mecanismos mais adequados para futuras aplicações foram definidas. Apesar de inicialmente planejada, a aplicação de uma metodologia de avaliação da segurança do protótipo construído não pôde ser realizada.

- **Integrar e adequar o protótipo a uma aplicação distribuída e híbrida, baseada em agentes móveis e agentes estacionários, de empresas virtuais**.

  O protótipo desenvolvido foi integrado a uma aplicação de empresas virtuais (EV) — sistema *MobiC-II*, contribuindo assim para a construção da confiança no processo de busca e seleção de parceiros para formação de uma EV e para contornar grande parte das ameaças de segurança encontradas no cenário definido para esta aplicação (cenário *TechMoldes*).

8.2 Principais Contribuições do Trabalho

Após o desenvolvimento do esquema de segurança proposto e a implementação do protótipo definido, as principais contribuições deste estudo de doutoramento podem ser assim identificadas:

- A utilização do SPKI/SDSI confere maior descentralização e escalabilidade ao esquema de segurança proposto do que a maioria dos mecanismos de proteção descritos na literatura. Apesar de usar algumas funcionalidades do modelo de segurança do Java, o esquema de autorização, baseado em certificados SPKI, garante o desacoplamento dos atributos de privilégios (credenciais) dos atributos de controle (políticas). Desta forma, o esquema ganha significativa flexibilidade com as possíveis delegações

oferecidas pelos certificados SPKI e maior dinamismo no processo de criação de domínios de proteção para execução de agentes.

- O autenticador *multi-hop* proposto baseia-se na autenticidade do proprietário do agente, no grau de confiança nas plataformas visitadas, na lista das federações definida pelo proprietário do agente e na própria integridade do agente. Devido a isto, o processo de autenticação se mostra mais confiável e adequado para agentes *multi-hop* e com itinerário livre do que as demais abordagens baseadas em informações relacionadas, exclusivamente, ao proprietário do agente.

- O MASS oferece ao desenvolvedor de aplicações de agentes móveis um conjunto de técnicas para proteção de agentes que podem ser selecionadas conforme os requisitos desejáveis de segurança de cada aplicação. Os repositórios de dados propostos são protegidos através de controles criptográficos que, além de protegerem a integridade e/ou confidencialidade, ligam os dados do agente à instância do mesmo (redundância criptográfica). Com isso, as técnicas propostas contornam algumas limitações e vulnerabilidades dos trabalhos correlatos, e ainda auxiliam o proprietário do agente e as plataformas visitadas a usarem e verificarem tais repositórios (através do autenticador *multi-hop*).

- O protótipo do *MASS* e a sua integração a uma aplicação real (Cenário *TechMoldes*) comprovam a flexibilidade e a viabilidade do conjunto de técnicas que compõem o esquema de segurança proposto para o uso em sistemas abertos.

Apesar das contribuições apresentadas, o esquema proposto nesta tese não contorna todas as ameaças de segurança presentes no paradigma de agentes móveis e por isso não se pode garantir que as aplicações baseadas em agentes móveis que utilizem este esquema estejam totalmente protegidas. Os perigosos ataques de plataformas maliciosas sobre a confidencialidade da computação do código dos agentes que podem levar a uma violação da propriedade intelectual não podem ser totalmente prevenidos e nem sempre detectados pelo esquema. Os ataques de plataformas maliciosas durante a computação de códigos é um problema universal que vem sendo tratado por alguns pesquisadores, porém uma solução efetiva e prática ainda não foi encontrada. Ataques de negação de serviço contra as plataformas de agentes podem ser minimizados com o uso dos domínios de proteção para execução de agentes móveis, porém alguns ataques deste tipo não podem ser evitados com o uso do *MASS*. Já os ataques de negação de serviço contra os agentes móveis, estes não são contornados com o *MASS*. O esquema proposto oferece alguns mecanismos que tratam da problemática da repudiação (objeto registrador de caminhos, autenticador *multi-hop* e mecanismo de controle social), porém estes não oferecem evidências suficientemente fortes para suportar a resolução dos desacordos ou para prevenir ataques de repudiação.

8.3 Resultados

Ao final das etapas realizadas e descritas anteriormente, obteve-se como resultado um protótipo que não só implementa boa parte dos mecanismos que compõem o esquema de segurança, como integra-se a uma aplicação de EV. Uma documentação detalhada descrevendo este protótipo encontra-se disponível no endereço `www.das.ufsc.br/AgenteSeg`.

De forma a difundir os conceitos e resultados obtidos com o esquema de segurança proposto e, principalmente, de submeter seus resultados para uma avaliação crítica da comunidade científica que se ocupa do problema da segurança em sistemas distribuídos, diversos documentos na forma de artigos e um mini-curso foram produzidos. As revisões e discussões provenientes que resultaram destas publicações contribuíram para elucidar algumas limitações e lacunas dos resultados preliminares e para motivar a superação destes problemas no esquema de segurança final (*MASS*). Conforme listados a seguir, os documentos científicos produzidos e publicados nesta tese foram: **um** artigo em periódico internacional, **um** artigo em periódico nacional, **quatro** artigos em eventos internacionais, **três** artigos em eventos nacionais e **um** mini-curso ministrado em um evento nacional.

- Wangham, M. S. ; Fraga, J.; Rabelo, R. J ; Lung, L. C. Secure Mobile Agent System and its Application in the Trust Building Process of Virtual Enterprises. Multiagent and Grid Systems, IOS Press - Amsterdam, v. 1, n. 3, p. 147-168, 2005.
- Wangham, M. S.; Fraga, J.; Schmidt, R.; Rabelo, R. J.. MASS: A Mobile Agent Security Scheme for the Creation of Virtual Enterprises. In: *First International Workshop on Mobility aware Technologies and Applications (MATA)*, Florianópolis. Springer-Verlag, Lecture Notes in Computer Science, v. v. 3284, p. 234-243. 2004.
- Wangham, M. S.; Fraga, J.; Obelheiro, R; Jung, G.; Fernandes, E.. Security Mechanisms for Mobile Agent Platforms Based on SPKI/SDSI Chains of Trust. In: Lucena, C.; Garcia, A.; Romanovsky, A.; Castro, J.; Alencar, P. (Eds.). *Software Engineering for Multi-Agents Systems II*. Nova York: Springer-Verlag, Lecture Notes in Computer Science, v. 2840, p. 207-224, 2004.
- Wangham, M. S.; Fraga, J.; Obelheiro, R. R.. A Security Scheme for Mobile Agent Platforms in Large-Scale Systems. In: *7th IFIP-TC6 TC11 International Conference on Communications and Multimedia Security*, Torino. Communications And Multimedia Security. Nova York: Springer-Verlag, Lecture Notes in Computer Science, v. 2828, p. 104-116, 2003.
- Rabelo, R. J; Wangham, M. S.; Schmidt, R.; Fraga, J.. Trust Building in the Creation of Virtual Enterprises in Mobile Agent based Architectures. In: *4th IFIP Working Conference on Virtual Enterprise*, Lugano. Processes and Foundations for Virtual, Organizations. Massachussets: Kluwer Academic Publishers, p. 65-72, 2003.

- Wangham, M. S.; Fraga, J.; Deitos, R.; Jung, G.. Repositórios Seguros de Dados para Proteção de Agentes Móveis contra Plataformas Maliciosas. In: *IV Workshop em Segurança de Sistemas Computacionais (WSeg)*, Gramado, 2004.

- Wangham, M. S.; Fraga, J.; Deitos, R.; Jung, G.. *Repositórios Seguros de Dados para Proteção de Agentes Móveis contra Plataformas Maliciosas*. Revista Scientia, Editora Unisinos, p. 83-03, 2004.

- Wangham, M. S.; Fraga, J.. Mecanismos de Segurança para Plataformas de Agentes Móveis, Baseados em Redes de Confiança SPKI/SDSI. In: *Simpósio Brasileiro de Redes de Computadores (SBRC)*, Natal, 2003.

- Wangham, M. S.; Fraga, J.; Santin, A. O.. Usando Certificados SPKI/SDSI para Geração de Domínios de Execução de Agentes Móveis. In: *Simpósio de Segurança em Informática (SSI)*, São José dos Campos, p. 111-120, 2002.

- Wangham, M. S.; Fraga, J.. Agentes Móveis X Segurança. 2001. (Apresentação de Mini-curso durante o Simpósio de Segurança em Informática - SSI´2001).

Os resultados obtidos com o esquema de segurança e sua implementação foram absorvidos em dois projetos de pesquisa: o **projeto Cadeias de Confiança** (CNPq/PROTEM - Conteúdos Digitais) e o **projeto Instituto Fábrica do Milênio** - IFM (MCT/CNPq), ambos desenvolvidos no Departamento de Automação e Sistemas da UFSC.

Espera-se que os resultados desta tese possam contribuir para uma melhor visão do problema da segurança em aplicações de agente móveis, motivem novos trabalhos nesta área e possam ser empregados em outros domínios de aplicação que anseiam por segurança.

8.4 Propostas de Trabalhos Futuros

Com o objetivo de analisar e explorar toda a potencialidade do *MASS*, sugere-se como continuidade deste trabalho a implementação do mecanismo de controle social proposto e da técnica para verificação das reputações das plataformas de agentes móveis, ambos baseados no conceito de Federações SPKI. Após concluído este refinamento do protótipo, a integração deste a outras aplicações de diferentes domínios poderá melhor avaliar a sua adequabilidade, eficiência e escalabilidade.

Como perspectivas futuras, pretende-se realizar um estudo de técnicas e mecanismos que visem prevenir e/ou tolerar ataques de negação de serviço voltados contra as plataformas ou contra os agentes móveis.

Outra proposta visaria cobrir uma preocupação de segurança que esteve fora do escopo desta tese — a proteção do código do agente contra a engenharia reversa maliciosa. Se

técnicas de prevenção forem consideradas não práticas, sugere-se assim desenvolver técnicas de segurança branda (*soft security*) que, ao menos, dificultem a dedução do comportamento de um agente pela análise de seu código, minimizando assim a ameaça contra a propriedade intelectual contida em um agente móvel.

Finalmente, a última proposta de trabalho futuro seria a aplicação de uma metodologia de testes para avaliar a segurança do protótipo (integrado à aplicação). Desta forma, alguns resultados poderiam ser melhor comprovados e refinamentos poderiam ser necessários para minimizar os riscos de segurança.

Apêndice A

Modelo de Segurança Java

A.1 Introdução

Desde sua introdução em 1995, a linguagem Java vem se tornando a mais popular das plataformas de desenvolvimento de aplicações (McGraw e Felten, 1999). O Java tem sido utilizado para criar páginas Web com conteúdo interativo e dinâmico, para desenvolver aplicativos corporativos de grande porte, para aprimorar as funcionalidades de servidores Web, fornecer aplicativos para dispositivos móveis, entre outras finalidades. Através dos seus *applets*, executados a partir de navegadores (*browsers* Web), a linguagem Java popularizou o conceito de códigos móveis (Thorn, 1997).

O ambiente de desenvolvimento da plataforma Java compreende três componentes principais: a **linguagem de programação** que produz, como resultado da compilação, um código intermediário e independente de arquitetura, chamado *bytecode*; a **máquina virtual Java** ou JVM (*Java Virtual Machine*) que interpreta o *bytecode* e atua como um sistema operacional portável para executar objetos (interpretador Java); um conjunto de classes ou **APIs** que compõem a plataforma Java, que são executadas sobre o interpretador fornecendo algumas classes básicas e úteis na construção de aplicações. A grande vantagem da interpretação é que a mesma permite que aplicações Java sejam executadas em qualquer sistema que possua uma implementação da máquina virtual Java (portabilidade).

Java é uma linguagem nova que foi elaborada para facilitar a criação de aplicações e de *applets* para estes serem usados em rede de computadores. As maiores preocupações dos criadores do Java foram torná-la, independente de plataforma, robusta, segura e fácil de ser trabalhada, usando o modelo cliente/servidor ou outras arquiteturas de rede.

Conforme apresentado no Capítulo 4, a plataforma Java, além de ser considerada um padrão de fato para programação de aplicações distribuídas, possui diversas características e propriedades que a tornam uma boa linguagem para programação de agentes móveis, tais como: interpretação segura de código, portabilidade, programação *multithread*, serialização

de objetos, reflexão estrutural, suporte para programação distribuída, carregamento dinâmico de código e assinatura de código (Lange, 1998a). Diante destas características, a linguagem Java e o seu modelo de segurança serviram de base para a concepção de diversos mecanismos e de esquemas de segurança, inclusive o descrito nesta tese. Este Apêndice tem por objetivo apresentar, de forma resumida, o modelo de segurança do Java e algumas de suas limitações.

A.2 Evolução do Modelo de Segurança

É fácil perceber que a possibilidade de criação de códigos executados localmente e compilados em qualquer lugar apresenta importantes implicações quanto à segurança. Estas implicações são constantemente consideradas como a principal limitação para o amplo uso do paradigma de códigos móveis, representado aqui pelos *applets* Java. Com o lançamento da Plataforma Java 2, as funcionalidades de segurança do Java tiveram um grande avanço e alguns dos problemas de segurança já foram contornados, porém algumas questões ainda precisam ser tratadas.

O modelo de segurança da plataforma Java, na sua proposição inicial, é conhecido como modelo *sandbox* (Gong, 1998). Este modelo fornece um ambiente restrito (uma "caixa de areia") para os códigos remotos (*applets*) que são considerados não confiáveis e podem acessar somente recursos limitados. Segundo este modelo, um código local é considerado confiável e tem acesso completo aos recursos do sistema, como o sistema de arquivos. Um gerenciador de segurança (*security manager*) é responsável por determinar quais acessos aos recursos são permitidos (Campione *et al.*, 2000). O JDK 1.0 (*Java Development Kit*) segue este modelo de segurança que vem sendo adotado pelas aplicações, incluindo os navegadores Web, habilitados a executar códigos Java. A Figura A.1 ilustra o modelo *sandbox* original.

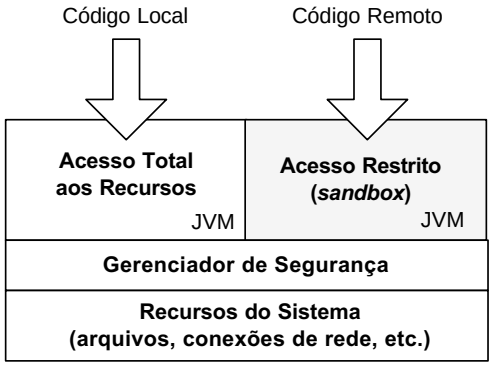


Figura A.1: Modelo de Segurança do JDK 1.0

O modelo de segurança do Java é aplicado através de uma variedade de mecanismos (Gong, 1998). Primeiramente, a linguagem foi projetada para ser segura[1] em relação aos tipos de dados utilizados e fácil de ser usada. Características como gerenciamento automático

[1]Trata-se aqui da segurança de funcionamento (*safety*).

de memória, coleta automática de lixo (*garbage collection*), acesso estruturado a memória, verificação dos limites de vetores, são exemplos de como a linguagem Java ajuda o programador a escrever códigos seguros. Compiladores e um verificador de *bytecode* garantem que somente *bytecodes* Java legítimos são executados. O verificador de *bytecode* inspeciona o código antes de sua execução sobre a máquina virtual. Um carregador de classes (*class loader*) define um espaço de nomes distintos para códigos não confiáveis para que estes não interfiram na execução de outros programas. O *class loader* recupera a classe correspondente, possivelmente de um sítio remoto e então carrega a classe na JVM. Nesse ponto o código é verificado e executado na JVM. A ligação de referências entre módulos de espaços de nomes é restrita a métodos públicos.

O modelo original, apresentado na versão 1.0, é bastante restrito e impede que programas bem intencionados, provenientes de fontes inseguras, executem uma determinada tarefa útil. O JDK 1.1 introduziu o conceito de *applet* assinado, como ilustrado na Figura A.2 , permitindo que um programa assinado por alguma entidade confiável pudesse ser tratado como código local, com acesso a todos os recursos. Segundo este modelo, ou deposita-se total confiança em uma classe ou nenhuma. Além do conceito de *applet* assinado, o JDK 1.1 definiu uma nova API chamada *Java Cryptography Architecture API* ou JCA, que se refere a um *framework* para acessar e desenvolver funcionalidades criptográficas para a plataforma Java. A API JCA inclui um fornecedor de serviço criptográfico ou CSP (*Cryptographic Service Provider*) que permite implementações criptográficas múltiplas e interoperáveis (Campione *et al.*, 2000).

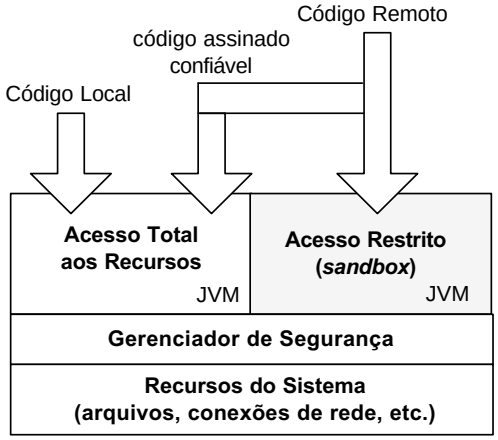


Figura A.2: Modelo de Segurança do JDK 1.1

Em 1998 foi lançado o Java 2. A arquitetura de segurança do JDK 1.2, ilustrada na Figura A.3, aspira aos seguintes objetivos (Gong, 1998): prover um controle de acesso de granularidade fina; possibilitar políticas de segurança facilmente configuráveis; e, definir uma estrutura de controle de acesso que pode ser estendida facilmente para todos os programas Java (códigos local e remoto). Todos os códigos, independentes, se local ou remoto, podem estar sujeitos a políticas de segurança. A política de segurança define um conjunto de permissões disponíveis para um código. Cada permissão especifica um acesso permitindo a

um recurso particular, como ler e escrever em um arquivo ou diretório específico (Campione *et al.*, 2000).

O conceito de domínio de proteção é fundamental no modelo de segurança do JDK 1.2. Um domínio pode ser definido como um conjunto de objetos que podem ser diretamente acessados por um principal. O principal é uma entidade do sistema cujas políticas definem as permissões concentradas em domínios, estabelecendo os seus acessos possíveis. O *sandbox*, utilizado no JDK 1.0, é exemplo de um domínio de proteção com limite fixo. Os domínios de segurança possuem geralmente duas categorias: domínios do sistema e domínios da aplicação. É importante que todos os recursos externos, tais como sistemas de arquivos, serviços de rede, tela e teclado estejam acessíveis através dos domínios do sistema.

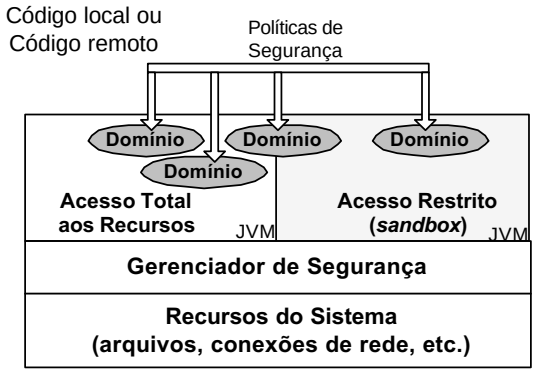


Figura A.3: Modelo de Segurança do JDK 1.2

O JDK 1.2 fornece ainda a API Java Cryptographic Extension (JCE) que estende o JDK para incluir APIs para cifragem, distribuição de chaves (suporte ao certificado X509v3) e funções para produzir *digests* de mensagens (MAC). O JDK1.2 fornece uma API criptográfica completa e independente da plataforma. O JCE é distribuído separadamente como uma extensão do JDK para estar de acordo com os regulamentos de controle de exportação dos EUA.

Além do JCE, a plataforma Java 2 conta ainda com outras duas funcionalidades de segurança opcionais para prover segurança às suas aplicações, a saber(Gong, 1998):

- *Java TM Authentication and Authorization Service* (JAAS): conjunto de pacotes que habilitam serviços de autenticação e aplicam controle de acesso sobre os usuários; implementam uma versão Java do *framework Pluggable Authentication Module* (Samar e Lai, 1996; Group, 1997), e fornecem suporte para controle de acesso baseado em papéis, em grupos e em usuários;
- *Java TM Secure Socket Extension* (JSSE): conjunto de pacotes que fornecem transferência segura de dados entre um cliente e um servidor rodando qualquer protocolo de aplicação (como HTTP, Telnet, FTP). O JSSE se executa sobre o TCP/IP, implementando uma versão Java dos protocolos SSL (*Secure Socket Layer*) e TLS (*Transport Layer Security*).

E, por fim, o JDK 1.2 fornece ainda três ferramentas: o *keytool*, usado para criar pares de chaves, para importar e mostrar cadeias de certificados e gerar certificados X.509v3 auto assinados; o *jarsigner* que assina arquivos JAR (Java Archive Format) e verifica a autenticidade das assinaturas dos arquivos assinados; e o *policy tool* que cria e modifica os arquivos de configuração de políticas.

Lançada em 13 de fevereiro de 2002, a versão mais atual da plataforma Java é a *Java TM 2 Plataform, Standard Edition v1.4*. Esta versão traz os seguintes melhoramentos ao modelo de segurança do Java 2 (Sun, 2002):

- Integração do JCE, JSSE e do JAAS ao Java 2 SDK (*Software Development Kit*);
- Duas novas funcionalidades de segurança:
  - a Java GSS-API que pode ser usada para trocar mensagens seguras entre duas aplicações que se comunicam usando o Kerberos v.5;
  - a Java *Certification Path* API que inclui novas classes e métodos que permitem construir e validar caminhos de certificados.
- A ferramenta *policy tool* foi aprimorada para permitir a especificação de principais, indicando para quais usuários as permissões de acesso estão sendo concedidas.

A.3 Limitações e Fraquezas do Modelo de Segurança

O modelo de controle de acesso do Java 2 possui algumas limitações que precisam ser analisadas. Ao invés de seguir a natureza distribuída do seu modelo de execução, o modelo de segurança do Java 2 está fundamentado em um esquema de autorização centralizado [2]. A aplicação do controle de acesso em tempo de execução está baseada no conceito de monitor de referência que pode ser implementado pela classe *SecurityManager*. Durante a sua execução, cada objeto é rotulado como pertencente a um domínio de proteção. Diferentes permissões são concedidas relacionando o par <domínio de proteção, recurso>. O grande problema é que, no Java 2, os direitos de acesso de cada componente (por exemplo, um *applet*) estão definidos em um arquivo de configuração no ambiente em tempo de execução. Este arquivo inclui o mapeamento entre cada componente móvel e as permissões concedidas a este componente para a sua execução no ambiente local. Além de numerosas dificuldades práticas em termos de programação, essas limitações impedem o desenvolvimento de um ambiente distribuído e dinâmico, exigindo uma definição estática de todos os componentes distribuídos e seus atributos de segurança de uma só vez.

[2] Essa centralização refere-se ao fato de que todo o controle de acesso é executado a partir de um arquivo único de configuração que define toda a política de segurança de uma máquina. Portanto, tem-se uma única ACL relacionando todos os sujeitos e objetos do sistema.

Outras limitações e fraquezas do modelo de segurança do Java estão relacionadas ao uso indiscriminado de recursos do sistema, como memória e CPU, que o Java permite. Como conseqüência destas, ataques de negação de serviço como os que visam consumir toda a memória disponível ou tornar o sistema extremamente lento (criando inúmeras *threads*) podem ser realizados. No Java, é extremamente difícil prevenir ou detectar este tipo de ataque já que o problema está em como diferenciar uma aplicação não maliciosa, que exige uma grande quantidade de memória (p. ex.: aplicações de geoprocessamento), de uma puramente maliciosa.

A.4 Conclusões

Atualmente a linguagem mais usada de desenvolvimento de aplicações distribuídas é a linguagem Java. A linguagem Java e o ambiente em tempo de execução garantem a segurança, primeiramente através de tipagem forte (Fuggetta *et al.*, 1998). A linguagem Java segue um modelo de segurança chamado de *sandbox* que permite executar códigos oriundos de fontes não confiáveis de forma isolada sem que estes realizem operações danosas ao funcionamento do sistema. Este modelo é usado para isolar a memória e o acesso aos métodos, e manter os domínios de execução, mutuamente exclusivos. A segurança é aplicada através de uma variedade de mecanismos. A verificação de *bytecode* é usada para checar a segurança de funcionalidade (*safety*) do código. Algumas verificações dinâmicas são também realizadas durante o tempo de execução. Um gerenciador de segurança media todos os acessos aos recursos do sistema, servindo como um monitor de referência.

A versão atual da plataforma Java 2 possui diversas funcionalidades de segurança que proporcionam aos programadores Java desenvolver aplicações com um nível aceitável de segurança. A especificação da arquitetura de segurança do Java aponta o gerenciamento de consumo de recursos, o agrupamento de permissões arbitrárias, a proteção em nível de objeto e a subdivisão de domínios de proteção como as direções para os próximos aprimoramentos da arquitetura de segurança.

Apêndice B

Segurança em Plataformas Comerciais e Acadêmicas de Agentes Móveis

B.1 Introdução

As plataformas de agentes móveis podem ser analisadas, em nível de linguagem, através de suas primitivas de programação de agentes. Nesta seção, destaca-se o interesse nas primitivas associadas com a segurança de agentes.

Como os agentes podem viajar por redes e sítios não confiáveis, o programador de agentes necessita de primitivas para garantir a confidencialidade dos dados (primitivas para cifragem e decifragem) e a integridade dos dados (primitiva para fornecer *digests*). Primitivas de assinaturas digitais e de verificação destas assinaturas também são necessárias para estabelecer comunicações entre pares autenticados. Se a criptografia de chave pública está sendo usada, primitivas para a geração de pares de chaves são necessárias, assim como uma infra-estrutura para gerenciar os possíveis certificados associados a estas chaves.

Sete plataformas comerciais e acadêmicas serão analisadas, considerando os mecanismos de segurança contra os agentes e as plataformas maliciosos, assim como os mecanismos empregados na proteção do canal de comunicação.

B.2 Aglets

B.2.1 Características Gerais

A API Java Aglets (J-AAPI)(IBM, 1996), desenvolvida pelo laboratório de pesquisa da IBM em Tóquio, Japão, se espelha no modelo de *applet* do Java e é, provavelmente, a mais famosa plataforma de agentes baseada na linguagem Java. A razão da popularidade está na

facilidade de usá-la. O termo *aglet* é a combinação das palavras *agent* e *applet*. Os agentes escritos com o Kit de Desenvolvimento de Software Aglets ou ASDK são chamados de *aglets*. Estas unidades de execução são *threads* que serão interpretadas por uma JVM (*Java Virtual Machine*). Recentemente a IBM disponibilizou o código fonte do sistema sob licença pública, aprovado como uma iniciativa de código aberto.

Os *aglets* são executados dentro de um *contexto-aglet* chamado de lugar na terminologia da OMG. Um *contexto-aglet* é o espaço de trabalho do agente e é responsável por manter e gerenciar os agentes em execução. Usando este contexto, um *aglet* pode obter referências de outros *aglets* (Gschwind, 2000). Para o carregamento de classes de um agente, o Aglets usa o seu próprio mecanismo. As classes podem ser carregadas de um repositório central ou do sítio anterior onde o *aglet* esteve. Visando reduzir a largura de banda, a plataforma Aglets facilita o armazenamento em *cache*, usando o seu próprio gerenciador de *cache*.

Uma característica interessante desta plataforma é o seu modelo de programação baseado em eventos *callback*. O sistema de agentes invoca métodos específicos sobre o agente quando certos eventos em seu ciclo de vida ocorrem. Por exemplo, quando um agente chega ao contexto-aglets destino, os métodos *onCreation* e *onArrival* são automaticamente invocados. O programador implementa uma classe de um agente, herdando implementações padrão desses métodos *callback* da classe Aglet e complementa-os com o código específico da aplicação (Karnik, 1998).

A migração de agentes no Aglets é absoluta, uma vez que esta requer a especificação da URL dependente da localização do sistema de agentes destino e suporta a abstração de itinerário. A mobilidade fraca é implementada usando serialização de objetos do Java, logo o estado de execução em nível de *thread* não é totalmente capturado. Os agentes são intermediados por objetos *proxy* que fornecem proteção em nível de linguagem, assim como localização transparente de agentes.

A comunicação entre agentes é realizada através de primitivas de passagem de mensagens (datagramas). Cada aglet pode possuir um método para tratar as mensagens que recebe de outros agentes. As trocas de mensagens podem ser síncronas, assíncronas (*one-way*) e *future-reply*. A plataforma usa nomes baseados nos nomes dos sítios e no número da porta e resolve estes nomes usando o *Domain Name System* ou DNS.

Com relação à confiabilidade, o Aglets tem uma característica instantânea que permite aos aglets serem verificados (através de primitivas de *checkpoint*) e reativados se estes forem acidentalmente terminados. As primitivas de *checkpoint* criam uma representação do estado do agente que pode ser armazenada em uma memória não volátil, para ser retomada quando necessário.

B.2.2 Aspectos de Segurança

Em Karjoth *et al.* (1997), integrantes do grupo de pesquisa da IBM apresentam um modelo de segurança para a plataforma Aglets, porém este modelo não se encontra totalmente implementado na versão corrente do Kit de Desenvolvimento de Software Aglets (ou ASDK). As seguintes funcionalidades de segurança são suportadas no ASDK1.1 (Oshima *et al.*, 1998):

- autenticação de usuários de um domínio;
- integridade da comunicação entre plataformas de um mesmo domínio;
- autorização com granularidade fina similar ao modelo de segurança do JDK1.2.

Um domínio no Aglets corresponde a um conjunto de contextos (chamado de lugares na terminologia da OMG) que seguem as mesmas políticas de segurança, conforme definida pela autoridade do domínio (Lange, 1998b) [1]. Todas as plataformas que pertencem a um mesmo domínio, compartilham uma chave secreta que pode ser usada na autenticação de plataformas quando contactadas, permitindo que as mesmas confirmem que pertencem ao mesmo domínio. Isto é realizado calculando um MAC (*Message Authentication Code*) sobre os dados a serem enviados, concatenados com um *nonce*. Depois que a autenticação entre as plataformas for estabelecida, as credenciais do *aglet* são enviadas junto com o agente. O receptor do agente irá então decidir o quanto ele confia nas credenciais com base nas informações obtidas na autenticação do sítio. No Aglets, as plataformas sempre confiam nas credenciais enviadas por plataformas do mesmo domínio, ou seja, a plataforma deposita confiança no *aglet* recebido, permitindo que o mesmo seja executado. Como todas as plataformas dentro de um domínio compartilham um segredo, a verificação da integridade da comunicação entre as plataformas pode ser realizada da mesma forma como foi descrita para a autenticação do domínio.

A autorização nesta plataforma é aplicada por uma implementação da interface *SecurityManager* do Java. Quando um *aglet* acessa informações sensíveis a segurança e a recursos, como as propriedades Java, *threads*, e/ou a qualquer outro recurso externo (arquivos), este acesso deve ser controlado, considerando as permissões dadas ao agente. As permissões podem ser especificadas por uma interface gráfica do usuário (GUI) ou podem editar diretamente a base de dados de políticas de segurança. O formato desta base de dados está de acordo com a especificação das permissões no JDK1.2.

Os *aglets* são identificados pelo seu *codebase* (identificação do código do agente) e pelo seu proprietário. O par (proprietário, codebase) é obtido do objeto *AgletInfo*, associado

[1] Políticas de segurança baseadas no domínio não estão implementadas na versão corrente do Aglets.

ao agente. As permissões são então definidas em termos do proprietário do agente e da informação do *codebase*.

No modelo de segurança implementado no ASDK, não há proteções contra plataformas maliciosas. A autenticação é realizada apenas para identificar se uma plataforma pertence a um domínio. Não é possível identificar e verificar os sítios de comunicação. Se a chave compartilhada for roubada de uma plataforma, não há como distinguir as plataformas válidas de uma plataforma que possui a chave roubada. A abordagem baseada em confiança, empregada no Aglets, não pode ser aplicada em redes abertas que contêm diversos domínios e plataformas. Usuários e proprietários de agentes não podem ser autenticados e suas identidades válidas com base na confiança depositada em uma plataforma. Os canais, por onde trafegam as mensagens e agentes, não possuem nenhuma proteção que garanta a confidencialidade da transmissão. Nenhum mecanismo de contabilização dos recursos do sistema é fornecido.

B.3 Concordia

B.3.1 Características Gerais

Desenvolvida em 1997 pela Mitsubishi Electric ITA, a plataforma Concordia, assim como muitos sistemas baseados em Java, suporta apenas mobilidade fraca, usando a serialização de objetos do Java e os seus mecanismos de carregamento de classes. A transferência de código pode ser feita sob demanda ou transferindo todas as classes necessárias de uma só vez. Cada objeto agente está associado com um objeto *Itinerary* que especifica o caminho da migração do agente (usando nomes DNS) e os métodos que serão executados em cada sítio. A versão corrente da plataforma, a 1.1.2, possui mais de 2Mb e contém a documentação e a GUI (Interface Gráfica do Usuário) pobres.

O Concordia apresenta um bom suporte à comunicação de agentes, fornecendo sinalização de eventos síncronos e assíncronos, mecanismos de colaboração em grupo (eventos *multicast*) e *broadcast* de eventos. Os nomes das entidades são dependentes da localização e baseados em DNS. O Concordia oferece ainda um Serviço de Diretório. As primitivas de *checkpoint*, suportadas pelo Concordia, verificam todo o movimento dos agentes e garantem uma confiabilidade excelente para as aplicações de agentes.

B.3.2 Aspectos de Segurança

O modelo de segurança do Concordia fornece três tipos de proteção (Walsh *et al.*, 1998): proteção do canal de comunicação para transferência e comunicação segura dos agentes; proteção dos sítios contra agentes maliciosos; e, proteção de um agente contra outros agentes. A

proteção do agente contra plataformas maliciosas não é tratada pelo Concordia. No Concordia, quando um sistema de agentes tiver sido identificado como uma plataforma Concordia válida, este é considerado um ambiente de execução confiável.

No sistema Concordia, o termo **proteção do agente** é usado para se referir à proteção de agentes contra modificações durante a transmissão ou quando estes estão armazenados em um meio persistente (disco rígido). A plataforma Concordia não implementa nenhuma funcionalidade adicional para proteger agentes enquanto residentes na memória, confiando esta tarefa às proteções oferecidas pela Máquina Virtual Java e pelo sistema operacional. Já o termo **proteção do recurso** se refere ao processo de proteção dos recursos da plataforma de acessos não autorizados. Esta proteção também é aplicada para proteger agentes de outros agentes maliciosos.

Para adicionar confiabilidade, o Concordia usa um gerenciador de objeto persistente que salva periodicamente o estado do agente. Como esta informação de estado é armazenada em disco, isto representa um risco à segurança do agente. Para evitar o acesso não autorizado, as informações do agente são cifradas usando algoritmos simétricos. A chave usada para esta cifragem é gerada aleatoriamente para cada agente recebido pelo servidor. Depois que a informação do agente é cifrada e escrita no disco, a chave simétrica usada é cifrada com a chave pública da plataforma e então é armazenada junto com o agente.

Os *bytecodes* de agentes não instanciados podem, opcionalmente, ser armazenados em arquivos JAR (Java Archive) assinados. O Concordia pode usar a assinatura sobre o arquivo JAR para garantir que o agente foi de fato escrito por um autor confiável e não foi modificado.

O Concordia oferece um canal de comunicação seguro usando o protocolo SSL v.3, fornecendo autenticação e serviços de cifragem para conexões TCP.

A abordagem para proteção dos recursos do sistema é construída sobre os mecanismos de segurança do Java que são estendidos para fornecer uma solução mais flexível para as plataformas de agentes móveis. Na filosofia do Concordia, a relação de confiança desejada não acontece com o autor do código mas sim com o proprietário do agente (pessoa pela qual o agente realiza a tarefa). Todo agente no Concordia está associado a um usuário particular através de uma identidade de usuário (Uto e Dahab, 2001). Esta identidade é um objeto Java composto pelo nome do usuário, o nome do grupo a qual este pertence e uma senha. A senha é sempre armazenada na forma de um *hash*.

Como o agente viaja pela rede, este carrega a sua identidade. Durante uma viagem do agente, sua identidade é protegida porque a senha está armazenada na forma de um *hash* e porque durante a viagem pela rede ou quando armazenada em disco, o agente está protegido por um canal seguro de comunicação. Em cada visita a uma plataforma, a identidade do agente é verificada de acordo com uma lista de usuários válidos do sistema. Cada plataforma é configurada com uma lista de usuários assim como com uma lista de permissões de acesso aos recursos. O sistema de agentes armazena sua lista de usuários em um arquivo chamado

arquivo de senhas. Este arquivo contém os nomes dos usuários e os valores *hash* das senhas. A verificação dos agentes que chegam é feita comparando o valor *hash* da senha que viaja junto com o agente com o valor armazenado no arquivo de senhas. Para previnir modificações não autorizadas, um outro *hash* é calculado considerando o arquivo de senhas e este pode ser digitalmente assinado pela plataforma.

Uma vez que o servidor identifica e valida o usuário, este servidor examina uma lista de permissões de recursos para saber qual o nível de acesso ao sistema é permitido ao usuário. Permissões de acessos a recursos são análogas às entradas de uma lista de controle de acesso ou ACL. Estas permissões podem ser definidas, pelo administrador do recurso, para usuários individuais ou para grupos de usuários. Permissões de recursos podem ser usadas para permitir ou negar acesso a recursos. As permissões se aplicam a todos os recursos controlados pela classe *SecurityManager* padrão do Java e a algumas funcionalidades adicionais como a criação e suspensão de agentes, o que obviamente determina a extensão da classe *SecurityManager*.

B.4 Grasshopper

B.4.1 Características Gerais

O Grasshopper (IKV++, 1999), desenvolvido pela GMD Fokus[2], é uma plataforma de agentes móveis, construída no topo de um ambiente de processamento distribuído. Desta forma, a integração do paradigma Cliente-Servidor tradicional com a tecnologia de agentes móveis pode ser facilmente encontrada.

O Grasshopper foi o primeiro sistema de agentes desenvolvido de acordo com o padrão proposto pela OMG, isto é, o *Mobile Agent System Interoperability Facility* - MASIF, atualmente chamado de MAF. A conformidade com o MASIF torna o Grasshopper muito atraente para pesquisadores da tecnologia de agentes móveis (IKV++, 1999).

A versão atual, a 2.1, possui uma boa documentação e uma boa GUI, porém, a plataforma é muito grande, mais de 4 Mb (sem as classes *Swing* e *OrbixWeb*). Assim como o Aglets, o Grasshopper é um sistema baseado em Java, com mobilidade fraca, e que usa o próprio carregador de classes para permitir que o agente carregue o código do agente de um *codebase* central ou da localização anterior do agente (Gschwind, 2000).

O serviço de comunicação do Grasshopper é responsável por todas as interações remotas entre os componentes distribuídos da plataforma, como também pelo transporte de agentes e pela localização de agentes por meio de um registro de região. Todas as interações podem ser executadas via CORBA IIOP, Java RMI(*Remote Method Invocation*) ou conexões planas

[2]Distribuído pela IKV -(*Informations und Kommunikations - Technologie Verbund*).

de sockets. O serviço de comunicação suporta comunicação síncrona, assíncrona, *multicast*, assim como invocação dinâmica de métodos (IKV++, 1999).

B.4.2 Aspectos de Segurança

O serviço de segurança do Grasshopper suporta dois tipos de mecanismos de segurança (IKV++, 1999): internos e externos. Os mecanismos de segurança externos protegem as interações remotas entre os componentes Grasshopper distribuídos. Estes mecanismos estão baseados no uso de certificados X.509 e no protocolo SSL (*Secure Socket Layer*). O SSL garante a confidencialidade (através de algoritmos simétricos), a integridade dos dados (através do MAC-*Message Authentication Code*) e a autenticação mútua entre as plataformas (utilizando certificados X.509). Como as seções SSL são estabelecidas entre os sistemas de agentes (chamados de agências), os proprietários de plataformas ou administradores de sistemas são autenticados, utilizando certificados X.509. Estes podem ser diferentes do programador do agente ou do ativador do mesmo.

Os mecanismos de segurança internos protegem os recursos das agências contra acessos não autorizados. O controle de acesso do Grasshopper é fortemente orientado pelos mecanismos de segurança do Java 2.[3]. Este faz uso de políticas de controle de acesso baseadas na identidade e no nome do grupo (*group membership*). No Grasshopper, uma política de controle de acesso, implementada no objeto *Policy*, consiste de uma lista de controle de acesso com as entradas associadas aos diversos sujeitos tratados na política. Um sujeito pode ter uma identidade única ou pertencer a um grupo. Um conjunto de permissões é associado a cada sujeito, concedendo a este o direito de acesso a recursos da plataforma Grasshopper.

Quando um agente tenta fazer um acesso a um recurso do sistema, um controlador de acesso é consultado para verificar se este é permitido. Na realidade, o controlador de acesso é invocado em toda requisição de acesso aos suportes de sistema. Este mecanismos é capaz de distinguir se o acesso foi feito por um agente ou por um código confiável do sistema, como por exemplo, o núcleo do Grasshopper. Se a tentativa de acesso vier de um agente, o controlador de acesso extrai a identidade do proprietário do agente (do próprio agente) e contata o objeto *Policy* que é uma representação em tempo de execução da política de controle de acesso do Grasshopper. Se o sujeito é membro de um ou mais grupos, as permissões do grupo são adicionadas às permissões individuais. Se o acesso for negado, uma exceção é levantada.

Quando um agente de uma agência remota faz um acesso a um sistema, a assinatura do agente é tomada como um *token* para buscar as permissões apropriadas de sua política de controle de acesso neste sistema. Esta assinatura é definida no tempo de criação do agente na sua agência e precisa ser protegida contra modificações.

[3]Porém, a implementação do *SecurityManager do Java* ainda está incompleta

Por questões de confiabilidade, o Grasshopper mantém uma cópia do objeto agente em um meio persistente (disco rígido). Entretanto, o Grasshopper não se preocupa com a segurança desta cópia armazenada. Também não há proteção dos agentes contra plataformas maliciosas.

B.5 Voyager

B.5.1 Características Gerais

Desenvolvido pela ObjectSpace (ObjectSpace, 1998) [4], o Voyager é uma plataforma para computação distribuída em Java que suporta agentes móveis com mobilidade fraca. Enquanto o Aglets e o Grasshopper são plataformas totalmente baseadas no paradigma de agentes móveis, o Voyager não foi projetado para ser somente uma plataforma de agentes. O Voyager combina as propriedades de um ORB, baseado em Java, com uma abordagem simples para transferência de agentes. Desta forma, o Voyager permite aos programadores criar aplicações, usando técnicas tradicionais de programação distribuídas e técnicas de agentes móveis (ObjectSpace, 1998). O ORB universal do Voyager combina técnicas de RMI, CORBA e DCOM. O Voyager visa uma programação distribuída fácil e transparente. É considerado um produto profissional e possui uma excelente documentação.

Devido aos custos e à dificuldade em transferir determinadas listas de classes, as classes dos agentes são usualmente carregadas sob demanda de um repositório central (Gschwind, 2000). Entretanto, um agente também pode determinar a forma como essas suas classes serão carregadas, fornecendo um objeto *Resource Loader* apropriado (ObjectSpace, 1998).

O Voyager oferece nomes independentes da localização em nível de aplicação. A plataforma fornece *proxies* locais para entidades remotas que encapsulam a sua localização corrente. O sistema atualiza a informação de localização no *proxy* quando as entidades se movem. Esta abordagem é usada para os nomes dos agentes, porém os nomes dos sistemas-de-agentes são identificados usando nomes DNS dependentes da localização. Um serviço de nomes é oferecido, permitindo uma localização transparente de um agente, a partir do seu nome ou identificador.

Quanto à comunicação entre as entidades do sistema, as facilidades de trocas de mensagens são enormes. O Voyager suporta comunicação síncrona, *one-way*, *future reply* e *multicast*, e, possui uma arquitetura para comunicação em grupo hierárquico (*broadcast* de eventos). E, por fim, a plataforma fornece interfaces transparentes e geração de *stubs* para um objeto, caso nenhuma interface seja explicitamente fornecida. Isto permite invocar remotamente métodos públicos de qualquer objeto, simplificando assim as interações entre objetos.

[4] A ObjectSpace foi comprada pela Recursion Software. A versão atual é a 4.1 e possui aproximadamente 700kb

O Voyager fornece seu próprio serviço de base de dados que pode ser usado para tornar os objetos persistentes. O movimento de agentes é considerado confiável devido às primitivas de *checkpoint* que este suporta.

B.5.2 Aspectos de Segurança

A plataforma Voyager fornece suporte opcional para comunicação segura entre agentes via adaptadores SSL e *tunneling* de *firewall*, usando o protocolo SOCKS. Nenhum mecanismo para proteção de agentes contra plataformas maliciosas e contra outros agentes é fornecido na versão 3.1 do Voyager (ObjectSpace, 99, 1998).

Para a proteção dos recursos da plataforma contra agentes maliciosos, o Voyager, baseado no modelo de segurança do Java 2, faz uso de um gerenciador de segurança chamado *VoyagerSecurityManager* (extensão do *SecurityManager* padrão) que pode ser opcionalmente instalado para restringir as operações que os agentes móveis podem executar. Neste modelo há somente duas categorias de segurança: nativo (código confiável) e externo (código não confiável). As operações sobre os lugares dos agentes e sobre os próprios agentes, como a criação e suspensão de agentes, não são controladas pelo gerenciador de segurança do Voyager. O programador do agente deve estender o *VoyagerSecurityManager* e incluir o controle a essas operações e a outras que não são tratadas pelo gerenciador.

B.6 Mole

B.6.1 Características Gerais

A plataforma de agentes móveis Mole (Baumann *et al.*, 1997, 1998) foi um trabalho de mais de cinco anos, desenvolvido desde 1996 no IPVR (*Institute for Parallel and Distribute Computer Systems*) da Universidade de Stuttgard, na Alemanha. A versão atual, 3.0, possui o código fonte disponível gratuitamente para uso não comercial, ocupa aproximadamente 360Kb e possui uma boa documentação, consistindo de manual de usuários, *Javadoc API* e diversos artigos [5].

Como a plataforma Mole utiliza a JVM padrão , esta suporta apenas mobilidade fraca. Já o suporte à mobilidade do Mole é implementado usando parte do pacote RMI e o processo de serialização fornecido no Java 1.1 (Baumann *et al.*, 1998). Quando o agente migra para um novo sítio, todas as classes utilizadas são transferidas de uma só vez para o sistema de agentes de destino. A plataforma não suporta movimento com segurança de funcionamento de agentes, mas a equipe do projeto tem diversos artigos sobre agentes tolerantes a falhas.

[5] A equipe do projeto Mole e as suas publicações atuam em quatro sub-áreas: controle de agentes, segurança de agentes, agentes tolerantes a falhas e aplicações de agentes móveis.

Com relação à comunicação entre as entidades do sistema, o Mole suporta RMI, passagem de mensagem e RPC orientados à sessão. No Mole, cada agente é identificado por um identificador único global. Este identificador, gerado pelo sistema no instante de criação do agente, é imutável. Esse esquema de identificadores fornece transparência de localização.

B.6.2 Aspectos de Segurança

A plataforma de agentes móveis Mole adota o modelo de segurança *sandbox*, em que todas as chamadas potencialmente perigosas não são permitidas, ou seja, os agentes não podem acessar diretamente nenhum recurso do sistema. O acesso aos recursos é feito de forma controlada e segura por intermédio de agentes estacionários (agentes de serviço) existentes nos sistemas de agentes (Baumann *et al.*, 1998). Um sistema de agentes pode especificar que tipo de agente este aceita executar, podendo este mecanismo ser usado para implementar restrições de acesso.

O Mole possui ainda um gerenciador de recursos, que protege a plataforma contra alguns ataques de negação de serviço. No Mole os seguintes recursos são gerenciados (Baumann *et al.*, 1998):

- tempo de uso da CPU;
- uso da rede;
- número de agentes-filhos criados;
- tempo total na plataforma.

O Mole não suporta mecanismos que garantam a proteção do canal de comunicação e proteção do agente contra plataformas maliciosas [6].

B.7 SOMA

B.7.1 Características Gerais

A plataforma de agentes móveis SOMA (*Secure Open Mobile Agent*) é um projeto que está sendo desenvolvido desde 1998 no DEIS da Universidade de Bologna, na Itália. O código binário e o código fonte do sistema estão disponíveis para uso não comercial. Esta

[6] Fritz Hohl, que propôs a técnica de caixa-preta limitada no tempo (Hohl, 1998), faz parte da equipe de pesquisa do projeto Mole. Apesar disso, não há documentos que apresentem os resultados da implementação desta técnica na plataforma Mole.

plataforma está baseada em Java (JDK2) e tem como objetivos principais a segurança e a interoperabilidade. Para atingir o objetivo da interoperabilidade, a plataforma SOMA é compatível com o CORBA e com a especificação MAF. Assim, um agente pode acessar objetos CORBA externos e operar com servidores CORBA. O SOMA pode ainda se tornar acessível a entidades externas através das interfaces MAF.

Além das características de segurança e interoperabilidade, o SOMA fornece transparência de localização, necessária para que o sistema tenha escalabilidade em um cenário Internet, e seja configurável e gerenciável dinamicamente via interfaces baseadas na Web. A nomeação associa entidades com identificadores únicos globais (GUID) e organiza estes identificadores em um sistema habilitado de nomes a localizar as entidades, mesmo que estas se movam. Esta facilidade permite a conformidade com diferentes sistemas de nomes (DNS, CORBA e LDAP).

O SOMA suporta mobilidade fraca através da primitiva *go*, disponível nos agentes. A transferência de código para o sistema de agentes destino é realizada sob demanda, conforme a necessidade do agente. A comunicação com um agente remoto é realizada através de primitivas de passagem de mensagens. Porém, a comunicação entre agentes, localizados no mesmo sistema de agentes, é possível através do compartilhamento de recursos, como um *buffer*. O SOMA suporta ainda primitivas que garantem a persistência de alguns objetos.

B.7.2 Aspectos de Segurança

A plataforma de agentes móveis SOMA foi projetada e implementada, considerando a segurança como uma propriedade chave. Esta plataforma leva em conta soluções de segurança padronizadas e usualmente empregadas em sistemas distribuídos e trata de várias questões de segurança que surgem em aplicações de agentes móveis (Bellavista *et al.*, 1999): a proteção dos sítios e dos agentes e a proteção dos canais de comunicação.

Uma infra-estrutura de confiança, baseada em políticas de segurança hierárquicas, é implementada em nível de domínio e em nível de lugar[7]. Um domínio define uma política de segurança global que impõe autorizações específicas ao mesmo; cada lugar lida com um conjunto de permissões definidas em nível de domínio. O SOMA protege os sítios contra agentes maliciosos, suportando autorização e autenticação de agentes em nível de domínio e de lugar.

Os agentes são autenticados com base em uma série de informações contidas na sua credencial. São elas: os nomes do domínio e do lugar de origem; o nome da classe que implementa o agente e o nome do usuário responsável pelo agente. Antes da migração, estas

[7] O sistema SOMA utiliza uma hierarquia de abstrações de localidade para organizar tarefas de gerenciamento e definir políticas. Em cada nó da rede existe pelo menos um lugar para execução de agentes. Diversos lugares são agrupados em domínios. Em cada domínio, um lugar padrão faz o papel de um *gateway* para comunicação entre domínios.

informações, o estado inicial do agente e o código do agente são digitalmente assinados pelo usuário que criou o agente. Quando o agente chega em um sítio remoto, suas credenciais são verificadas em relação à sua autenticidade (assinatura do proprietário do agente).

Uma vez autenticados, os agentes estão autorizados a interagir com os recursos do lugar de acordo com a política de segurança corrente. Assim, principais são associados com papéis (*roles*) que identificam as operações permitidas sobre os recursos do sistema. Cada plataforma tem as suas listas de controle de acesso baseadas em papéis (RBAC). Uma ferramenta gráfica para o gerenciamento dos papéis em tempo de execução está disponível em cada plataforma SOMA. Os agentes podem ainda ser autorizados com base em suas preferências (um conjunto de credenciais definidas pelo proprietário do agente). Os mecanismos do controle de acesso do SOMA são modelados de acordo com o conceito de domínios de proteção do Java. Agentes podem apenas executar operações e acessos aos recursos especificados dentro do seu domínio de proteção. O SOMA usa o mecanismo de carregamento seguro de classes do Java para associar agentes com um domínio de proteção correspondente.

A proteção do agente durante a sua transferência ocorre devido ao uso de canais autenticados e cifrados. Antes de ser transferido, o agente é cifrado e assinado digitalmente. O lugar de destino aceita o agente somente se este confia e verifica a autenticidade das credenciais. É possível escolher entre um canal cifrado com DES ou uma solução baseada no protocolo SSL.

O SOMA também protege os agentes contra o mau-comportamento do ambiente de execução, adotando protocolos para preservar a integridade do estado do agente. Duas soluções são apresentadas em Corradi *et al.* (1999b): uma baseada em uma terceira parte confiável (TTP) que valida os dados coletados pelo agente, e a outra baseada em um protocolo distribuído chamado de *Multiple-Hop* (MH) que dispensa o uso de uma terceira parte confiável. Estas duas técnicas foram analisadas no Capítulo 6 (seção 6.2.1).

B.8 Ajanta

B.8.1 Características Gerais

O projeto Ajanta (Karnik, 1998), iniciado em 1997, está sendo desenvolvido na Universidade de Minnesota, nos EUA. Esta plataforma de agentes móveis está baseada na linguagem Java (1.1.5) e o seu código binário está disponível gratuitamente para uso não comercial. O Ajanta tem como objetivo básico de projeto contornar os problemas de segurança quando agentes móveis fazem parte de um sistema.

A arquitetura Ajanta inclui um sistema de agentes (chamado servidor de agentes) que fornece um ambiente de execução seguro, utiliza um protocolo de transferência de agentes

(ATP) que protege a integridade e a confidencialidade do agente e usa a serialização de objetos padrão do Java para capturar parte do estado de execução para a migração (mobilidade fraca). A transferência do código, necessária à execução do agente é realizada sob demanda, a partir do sistema originário de agentes. O Ajanta também tem um suporte à abstração de itinerários.

Qualquer aplicação que utiliza a plataforma Ajanta cria um objeto *guardian* responsável por tratar todas as exceções não resolvidas pelos próprios agentes da aplicação, proporcionando assim uma maior confiabilidade.

No Ajanta, as entidades que compõem a plataforma recebem nomes globais independentes da localização, baseados em URNs (*Uniform Resource Names*)(Moats, 1997). Um serviço de nomes foi implementado para resolver as URNs.

A comunicação entre agentes de um mesmo sistema de agentes é feita através da invocação de métodos ou do uso de recursos compartilhados. No caso da invocação de métodos, um agente deve se registrar como um recurso de um servidor e fornecer *proxies* para a comunicação com os outros agentes. Para a comunicação com agentes externos ao seu sistema de agentes, o Ajanta suporta ainda RMI.

B.8.2 Aspectos de Segurança

O plataforma de agentes móveis Ajanta oferece mecanismos para proteção do agente, para a proteção da plataforma e para a comunicação segura (Karnik, 1998).

O serviço de nomes do Ajanta também tem como função manter uma infra-estrutura para distribuição de chaves públicas. Estas chaves públicas são usadas para assinaturas digitais (protocolo DSA) e para cifragem do canal de comunicação (El-Gamal) e são associadas ao URN (*Uniform Resource Name*) de cada entidade.

O protocolo para autenticação foi desenvolvido usando um mecanismo de desafio-resposta (*challenge-response*), com geração aleatória de *nonces* para prevenir ataques de mensagem antiga. Este opera em nível de aplicação e permite autenticação mútua dos URNs. As entidades autenticadas podem ser os agentes, seus proprietários, os sistemas de agentes, os resolvedores de nomes, etc.

A plataforma Ajanta possui um protocolo seguro para transferência de um agente chamado ATP (*Agent Transfer Protocol*). Este protocolo incorpora autenticação mútua dos sistemas de agentes e cifragem dos dados para garantir a confidencialidade e a integridade dos dados do agente. No último passo do protocolo, o sistema originário de agentes atualiza a localização do agente no Serviço de Nomes.

A plataforma Ajanta, que incorpora um ambiente seguro de execução para os agentes, usa as facilidades do Java para implementar domínios de proteção para os agentes. Estes

domínios isolam os agentes, prevenindo que agentes maliciosos modifiquem outros agentes.

O acesso aos recursos do sistema é controlado pelo gerenciador de segurança da plataforma Ajanta, que estende o *RMISecurityManager* do Java[8], e que utiliza listas de controle de acesso, baseadas nos URNs dos usuários. Para a proteção dos recursos da aplicação, esta plataforma usa um mecanismo baseado no conceito de interposição de *proxy* entre o recurso e o cliente, e permite aos desenvolvedores do recurso da aplicação implementar sua própria política.

A plataforma Ajanta fornece ainda três mecanismos para proteger os agentes contra plataformas maliciosas. Estes mecanismos permitem detectar modificações indesejadas no estado do agente (Karnik, 1998). Estes três mecanismos foram apresentados no Capítulo 6 (seções 6.2.1 e 6.2.2).

B.9 Considerações Finais

Todas as plataformas analisadas estão baseadas na linguagem Java. Como nenhuma das plataformas modifica a máquina virtual para obter todo o estado de execução, estas suportam apenas mobilidade fraca. A tabela B.1 resume e compara as plataformas, considerando as seguintes características: nomeação, comunicação, primitivas de tolerância a falhas, primitivas de controle e gerenciamento de agentes.

Entre as plataformas que não são baseadas em Java e que merecem destaque, podem-se citar as plataformas ARA, Telescript e D´Agents. Historicamente, a plataforma Telescript da General Magic (Tardo e Valente, 1996) é muito importante, pois foi a primeira plataforma comercial de agentes móveis. Porém, como esta requeria o aprendizado de uma nova linguagem, foi comercialmente mal sucedida. Esta plataforma suporta mobilidade forte, está baseada em uma linguagem orientada a objetos "tipo-segura"e possui mecanismos para contornar algumas ameaças de segurança encontradas no paradigma de agentes móveis. A plataforma ARA (*Agents for Remote Action*) (Deine e Stolpmann, 1997), desenvolvida na Universidade de Kaiserlautern, teve por objetivo principal adicionar mobilidade a algumas linguagens existentes de programação. A plataforma ARA suporta as linguagens Tcl, C e C++; as plataformas Sparc Solaris, Intel Linux e Sparc SunOS, bem como suporta mobilidade forte em todas as linguagens e primitivas de segurança e tolerância a falhas. Antigamente denominada de Agent Tcl, a platforma D´Agents (Gray, 1996; Gray *et al.*, 1998) vem sendo desenvolvida no Dartmouth College com o apoio de diversas entidades como o DARPA (*Defense Advanced Research Projects Agency*) e o *Office of Naval Research* dos E.U.A. Os agentes desta plataforma podem ser escritos em Tcl, Java e Scheme, e, possuem mobilidade forte. A plataforma D´Agent suporta boas primitivas de comunicação e sincronização agente-agente e primitivas de segurança.

[8]Classe que fornece uma implementação do gerenciador de segurança adequado para o controle de acesso de códigos remotos.

Plataformas	**Nomeação**	**Comunicação**	**Tolerânica a Falhas**	**Controle Monitor.**
Aglets	dependente da localização	passagem de mensagem, síncrona *oneway, future reply*	sim	sim
Concordia	dependente da localização	invocação de método local síncrono, assíncrono e *multicast*	sim	não
Grasshopper	dependende da localização	CORBAIIOP, RMI, sockets, síncrona, assínc., *multicast* e invoc. dinâmica	sim	sim
Voyager	independente da localização	RMI, CORBA e DCOM, sínc. *one-way, futere-reply* e *multicast*	sim	sim
Mole	independente da localização	RMI, passagem de mensagens e RPC síncronas e assíncronas	não implement.	não implement.
MOA	independente da localização	canal de comunicação (topo dos sockets), síncronas e assíncronas	sim sim	sim sim
SOMA	independente da localização	passagem de mensagem e *buffer*	sim	sim
Ajanta	independente da localização	invoc. de método local via *proxy* e RMI via *proxy*	sim	sim

Tabela B.1: Comparação das Plataformas Analisadas

A Tabela B.2, a seguir, resume as características de segurança das plataformas, analisando os seguintes requisitos:

- mecanismos de proteção da plataforma contra ataques de outros agentes;
- mecanismos de proteção da plataforma contra ataques de outras plataformas;
- mecanismos de proteção dos agentes contra ataques de outros agentes;
- mecanismos de proteção dos agentes contra ataques de plataformas;
- mecanismos de proteção do canal de comunicação.

Todas as plataformas analisadas se utilizam do modelo de segurança do Java, em especial o do Java 2, para implementar seus mecanismos de segurança. A maioria das plataformas estende o gerenciador de segurança do Java para fornecer uma solução mais flexível e adequada para as plataformas de agentes e para implementar domínios de proteção que isolam os agentes móveis prevenindo que ataques maliciosos dos mesmos ocorram. Em algumas plataformas, a diferença dos esquemas de autorização está na informação usada para determinar o conjunto de direitos de acesso de um agente, pois a maioria usa somente a identidade do proprietário do agente, enquanto outras usam o grupo a que este pertence ou o papel atribuído ao mesmo (ver Tabela B.2). Para a proteção contra outras plataformas, a solução empregada pela maioria das plataformas foi a autenticação mútua, utilizando o protocolo SSL como tecnologia de segurança.

A comunicação remota e a transferência de agentes estão protegidas contra a revelação e a modificação não autorizada de dados através de controles criptográficos implantados nas infra-estruturas de comunicação de quase todas as plataformas de agentes (ver Tabela B.2).

Analisando a Tabela B.2, observa-se que a segurança dos agentes contra as plataformas maliciosas é um problema ainda não resolvido. Somente as plataformas SOMA e Ajanta procuram detectar, e não evitar, os perigosos ataques provenientes de plataformas maliciosas. Para garantir a integridade do código do agente, ambas utilizam técnicas de assinatura digital. Quanto à proteção do estado do agente, a plataforma SOMA utiliza técnicas de detecção (protocolos TTP e MH) que possuem diversas limitações, já as técnicas de detecção da plataforma Ajanta se mostram mais adequadas para este tipo de proteção. Porém, nenhuma das plataformas garante a confidencialidade do código do agente, contendo, por exemplo, estratégias de negociação, durante a interpretação do código.

Plataformas	Mecanismos de Proteção do Agente		Mecanismos de Proteção da Plataforma		Canal Seguro
	contra outros Agentes	contra a Plataforma	contra os Agentes	contra outras plataformas	
Aglets	Domínios de proteção do Java	Não suporta	Estende o *SecurityManager*. Autorização baseada no par *codebase* e proprietário.	Autenticação das plataformas pertencentes ao seu domínio	Autenticação e integridade (MAC)
Concordia	Domínios de proteção do Java	Não suporta	Estende o *SecurityManager*. Autorização baseada na identidade do usuário responsável pelo agente	Autenticação mútua (protocolo SSL)	Autenticação e confidencialidade via SSL
Grasshopper	Domínios de proteção do Java	Não suporta	Estende o *SecurityManager*. Autorização baseada na identidade do proprietário do agente e no grupo que este pertence	Autenticação mútua (protocolo SSL)	Autenticação e confidencialidade via SSL
Voyager	Não suporta	Não suporta	Autorização baseada em duas categorias: nativo ou externo. O programador pode estender o gerenciador de segurança	Autenticação das plataformas (opcional)	Autenticação e confidencialidade via SSL (pacote opcional)
Mole	Domínios de proteção do Java	Não suporta	Gerenciador de recursos. Autorização baseada no modelo *sandbox*. Acesso mediado por agentes estacionários.	Não suporta	Não suporta
SOMA	Domínios de proteção do Java	Integridade do código: assinatura digital. Integridade do agente: protocolos TTP e MH	Estende o *SecurityManager*. Autorização baseada em papéis associados aos usuários (proprietários)	Autenticação das plataformas	Confidencialidade (DES) ou autenticação e confidencialiade via SSL
Ajanta	Domínios de proteção do Java	Mecanismos para detectar modificações do código e do estado do agente	Estende o *SecurityManager*. Autorização baseada no proprietário do agente. Proteção dos recursos: interposição de um *proxy* mediador de acessos	Autenticação das plataformas	Confidencialidade (El Gamal) e autenticação via DSA

Tabela B.2: Tabela Comparativa das Funcionalidades de Segurança das Plataformas de Agentes Móveis

Referências Bibliográficas

Ad Astra Engineering, I. (1998). *Jumping Beans White Paper*.

Amoroso, E. G. (1994). *Fundamentals of Computer Security Technology*. Prentice Hall PTR.

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., e Yang, K. (2001). On the (im)possibility of obfuscating programs. Em Kilian, J., editor, *Advances in Cryptology-CRYPTO 2001*.

Baumann, J., Hohl, F., Rothernel, K., Schwehn, M., e Straber, M. (1998). Mole 3.0: A middleware for java-bases mobile software agents. Proc. *Middleware'98*. Springer.

Baumann, J., Hohl, F., Rothernel, K., e Straber, M. (1997). Mole - concepts of a mobile agent system. Relatório técnico, University of Stuttgart.

Bellavista, P., Corradi, A., e Stefanelli, C. (1999). A secure and open mobile agent programming environment. Proc. *Proceending of the Fourth International Symposium on Autonomous Decentralized Systems*, pp. 238–245. IEEE Computer Society Press.

Bishop, M. (2003). *Computer Security*. Addison Wesley.

Blaze, M., Feigenbaum, J., e Lacy, J. (1996). Decentralized trust management. Proc. *The 17th IEEE Symposium on Security and Privacy, IEEE Computer Society*.

Bray, T., Paoli, J., e Sperberg-McQueen, C. M. (1998). Extensible markup language (xml) 1.0 - w3c recommendation,. Relatório Técnico REC-xml-19980210, W3C, http://citeseer.nj.nec.com/bray98extensible.html.

Breugst, M. e Magedanz, T. (1998a). 5th international conference on intelligence in services and networks. Em Trigila, S., editor, *On the Usage of Standard Mobile Agent Platforms in Telecommunication Environments*, pp. 275–286. Springer.

Breugst, M. e Magedanz, T. (1998b). Mobile agents - enabling technology for active intelligent network implementation. *IEEE Network Magazine*, Vol. 12, No. 3, pp. 53–60.

Camarinha-Matos, L. e Afsarmanesh, H. (1999). The ve concept. Em Camarinha-Matos, L. e Afsarmanesh, H., editors, *Infrastructures for VE - Networking Industrial Enterprises*, pp. 3–14. Kluwer Academics Publishers.

Camarinha-Matos, L. e Afsarmanesh, H. (2002). Dynamic virtual organizations, or not so dynamic ? Proc. *Third IFIP Working Conference on Virtual Enterprise (PRO-VE´2002)*, pp. 111–124.

Campione, M., Walrath, K., e Huml, A. (2000). *The Java Tutorial*. Addison Wesley, third edition edição. http://java.sun.com/docs/books/tutorial.

Carzaniga, A., Picco, G. P., e Vigna, G. (1997). Designing distributed applications with mobile code paradigms. Proc. *In Proceedings of the 1997 International Conference on Software Engineering*.

Chess, D. M. (1998). Security issues in mobile code systems. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*. Springer.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., e Tsudik, G. (1995). Itinerant agents for mobile computing. *IEEE Personal Communications*, Vol. 2, No. 5, pp. 34–49.

Clarke, D. E. (2001). Spki/sdsi http server/certificate chain discovery in spki/sdsi. Dissertação de Mestrado, Massachusetts Institute of Technology - MIT.

Clark, J. e DeRose, S. (1999). Xml path language (xpath) version 1.0 - w3c recommendation. Relatório Técnico REC-xpath-19991116, W3C, http://www.w3.org/TR/xpath.

Colby, C., Lee, P., e Necula, G. C. (2000). A proof-carrying code architecture for java. Em Emerson, E. A. e Sistla, A. P., editors, *Proceedings of the 12th Internation Conference of Computer Aided Verification (CAV)*.

Collberg, C., Thomborson, C., e Low, D. (1997). A taxonomy of obfuscating transformations. Relatório Técnico 148, Department of Computer Science, University of Auckland.

Corradi, A., Cremonini, M., Montanari, R., e Stefanelli, C. (1999a). Mobile agents integrity for electronic commerce applications. *Information Systems*, Vol. 24, No. 6, pp. 519–533.

Corradi, A., Montanari, R., e Stefanelli, C. (1999b). Security issues in mobile agent technology. Proc. *7th IEEE Workshop on Futere Trends of Distributed Computing Systems*.

Couloris, G. e Dollimore, J. (2001). *Distributed Systems - Concepts and Design*. Addison-Wesley.

CSI/FBI (1999). Csi/fbi computer crime and security survey. Computer Security Institute Publication. Computer Security Institute and Federal Bureau of Investigation.

Dasgupta, P., Narasimhan, N., Moser, L. E., e Melliar-Smith, P. M. (1998). A supplier driven electronic marketplace using mobile agents. Proc. *First International Conference on Telecommunications and E-Commerce*.

Deine, H. e Stolpmann, T. (1997). The architecture of ara plataform for mobile agents. Proc. *In First International Workshop on Mobile Agent MA´97*.

Denning, D. (1982). *Criptography and Data Security*. Addison-Wesley.

DoD (1985). Department of defense trusted computer system evaluation criteria. DoD 5200.28-STD.

dos Santos, I. S. (2004). *Dai-lhes vós mesmos de comer*.

Elien, J. (1998). *Certificate Discoveru Using SPKI/SDSI 2.0 Certificates*. Tese de Doutorado, Massachusetts Institute of Technology MIT.

Elisson, C. (1999). *SPKI Requirements (RFC 2692)*. The Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2692.txt.

Farmer, W., Guttman, J., e Swarup, V. (1996a). Security for mobile agents: Authentication and state appraisal. Proc. *4th European Symposium on Research in Computer Security (ESORICS'96)*.

Farmer, W., Guttman, J., e Swarup, V. (1996b). Security for mobile agents: Issues and requirements. Proc. *Proc. 19th National Information System Security Conference*.

FIPA (2000). Fipa agent management specification. FIPA00023 Foundation for Intelligent Physical Agents.

Ford, W. (1994). *Communications Security: Principles, Standard Protocols and Techniques*. Prentice-Hall.

Franklin, S. e Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. Proc. *Third International Workshop on Agent Theories, Architectures, and Languages*. Springer.

Freier, A., Karlton, P., e Kocher, P. (1996). Secure socket layer 3.0. Internet Draft-IETF. http://wp.netscape.com/eng/ssl3/ssl-toc.html.

Fuggetta, A., Picco, G. P., e Vigna, G. (1998). Understanding code mobility. *Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342–361.

Garfinkel, S. e Spafford, G. (1997). *Web Security Commerce*. O'Reilly Associates Inc.

Gong, L. (1998). Java 2 plataform security architecture. http://java.sun.com.

Gray, R. S., Kotz, D., Cybenko, G., e Rus, D. (2000). Mobile agents: Motivations and state-of-the-art systems. Relatório Técnico TR2000-365, Dartmouth College, Computer Science.

Gray, R. S. (1996). Agent tcl: A flexible and secure mobile-agent system. Proc. *Fourth Annual USENIX Tcl/Tk Workshop*, pp. 9–23.

Gray, R., D.Kotz, Cybenko, G., e Rus, D. (1998). D´ agents: Security in a multiple-language, mobile agent systems. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419. Springer.

Group, T. O. (1997). X/open single sign-on service (xsso)- pluggable authentication. Preliminary Specification P702.

Gschwind, T. (2000). Comparing object oriented mobile agent systems. Proc. *6 th ECOOP 2000-Workshop on Mobile Object Systems*.

Haggar, P. (2000). *Java: Guia Prático de Programação*. Campus.

Hohl, F. (1998). Time limited blackbox security: Protecting mobile agents from malicious hosts. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*, pp. 92–113. Springer.

IBM (1996). Aglets software development kit. http://www.trl.ibm.co.jp/aglets.

IEFT (1999). Kerberos network authentication service - versão 5. RFC2704. http://www.ietf.org/rfc/rfc2704.txt.

IKV++ (1999). Grasshopper 2: The agent plataform. http://www.ikv.de/products/grasshopper2.index.html.

Islam, N., Anand, R., Jaeger, T., e Rao, J. R. (1997). A flexible security system for using internet content. *IEEE Softw.*, Vol. 14, No. 5, pp. 52–59.

ISO/IEC (1999). Iso/iec 15408-1:1999(e)- part i: Introduction and general model. ISO/IEC 15408.

ITU-T (1993). Itu-t- information technology - the open systems interconnection - the directory: Authentication framework. ITU-T Recomendation X.509.

Jalote, P. (1991). *An Integrated Approach to Software Engineering*. Springer-Verlag.

Jansen, W. A. (2000). Countermeasures for mobile agent security. *Computer Communications*, Vol. 23, No. 17, .

Jansen, W. e Karygiannis, T. (1999). Mobile agent security. Relatório Técnico NIST Special Publication 800-19, National Institute osf Standards and Technology.

Karjoth, G., Asokan, N., e C.Gülcü (1998). Protecting the computing results of free-roaming agents. Proc. *Proc. of the Second International Workshop on Mobile Agents*.

Karjoth, G., Lange, D., e Oshima, M. (1997). A security model for aglets. *IEEE Internet Computing*, Vol. 1, No. 4, pp. 68–77.

Karnik, N. e Tripathi, A. (2001). Security in the ajanta mobile agent system. *Software—Practice and Experience*, Vol. 31, No. 4, pp. 301–329.

Karnik, N. (1998). *Security in Mobile Agent System*. Tese de Doutorado, University of Minnesota. http://www.cs.umn.edu/Ajanta.

Knudsen, P. (1995). Comparing two distributed computing paradigms - a performance case study. Dissertação de Mestrado, University of Tromsø.

Kotz, D., Gray, R. S., e Rus, D. (2002). Mobile agents: Future directions for mobile agent research. *IEEE Distributed Systems Online*, Vol. 3, No. 8, .

Lampson, B. W. (1971). Protection. Proc. *5 th Princeton sysmposium on Information Sciences and Systems*.

Lampson, B. W. (1973). A note on the confinement problem. *Communication of the ACM*, Vol. v.16, No. n.10, pp. 613–615.

Lampson, B. e Rivest, R. (1996). A simple distributed security infrastructure. http://theory.lcs.mit.edu/ cis/sdsi.html.

Landi, W. (1992). Undecidability of static analysis. *ACM Leters on Programming Languages and Systems*, Vol. 1, No. 4, pp. 323–337.

Landwehr, C. E. (1981). Formal models for computer security. *Computing Surveys ACM*, Vol. 13, No. 3, .

Landwehr, C. E. (2001). Computer security. *International Journal of Information Security*, Vol. 1, No. 1, pp. 3–16.

Lange, D. B. (1998a). Mobile objects and mobile agents: The future of distributed computing. Proc. *European Conference on Object-Oriented Programming'98*.

Lange, D. (1998b). Mobile agents with java: The aglet api. *World Wide Web Journal*, Vol. 1, No. 3, .

Levy, J. Y., Ousterhout, J. K., e Welch, B. B. (1997). The safe-tcl security model. Relatório Técnico SMLI TR-97-60, Sun Microsystems.

Loureiro, S. e Molva, R. (1999). Fuction hiding based on error correcting codes. Em Blum, M. e Lee, C., editors, *Proc. of Cryptec´99- International Workkshop on Cryptographic Techniques and Electronic Commerce*, pp. 92–98.

Magedanz, T., Rothermel, K., e Krause, S. (1996). Intelligent agents: An emerging technology for next generation telecommunications ? Proc. *INFOCOM'96*, San Francisco, CA, USA.

Magedanz, T. (1999). Omg and fipa standardisation for agent technology: competition or convergence? http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch2/ch2.htm.

McGraw, G. e Felten, E. (1999). *Securing Java*. John Wiley and Sons, Inc.

Mello, E. R. (2003). Redes de confiança em sistemas de objetos corba. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Moats, R. (1997). *RFC2141: URN Syntax*. http://www.ietf.org/rfc/rfc2141.txt.

Molva, R. e Roudier, Y. (2000). A distributed access control model for java. Proc. *European Symposium on Research in Computer Security - ESORICS 2000*.

Morcos, A. (1998). A java implementation of simple distributed security infrastructure. Dissertação de Mestrado, Massachusetts Institute of Technology.

Necula, G. e Lee, P. (1998). Safe, untrusted agents using proof-carrying code. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*, pp. 61–91. Springer.

Neuman, B. C. e TS'o, T. (1994). Kerberos: An authentication service for computer network. *IEEE Communications*, Vol. 32, No. 9, pp. 33–38.

Neuman, B. C. (1994). *Readings in Distributed Systems*, chapter Scale in Distributed Systems. ISBN 0818630329. IEEE Computer Society Press.

Nicomette, V. (1996). *La Protection dans les Systèmes à Objets Répartis*. Tese de Doutorado, Institut National Polytechnique de Toulouse.

Nikander, P. e Partanen, J. (1999). Distributed policy management for jdk 1.2. Proc. *Proceedings of the 1999 Network and Distributed Systems Security Symposium*. Internet Society.

NIST (1997). *Entity Authentication Using Public Key Cryptography*. National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips196/fips196.pdf.

NIST (2000). Minimum interoperability specifications for pki components, version 2- second draft. National Institute of Standards and Technology.

Obelheiro, R. R. (2001). Modelos de segurança baseado em papéis para sistemas de larga escala: a proposta rbac-jacoweb. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

ObjectSpace (1998). Objectspace voyager 2.0. http://www.objectspace.com/products/voyager1.htm.

ObjectSpace (99). Voyager orb 3.1 developer guide. Relatório técnico, ObjectSpace.

Ogiso, T., Sakabe, Y., Soshi, M., e Miyaaji, A. (2003). Software obfuscation on a theoretical basis and its implementation. *IEICE Transaction on Fundamentals*, Vol. E86-A, No. 1, pp. 176–186.

OMG (2000). Mobile agent facility specification. OMG Document 2000-01-02.

OMG (2004). The common object request broker architecture v3.0.3. OMG Document 04-03-01.

Ordille, J. J. (1996). When agents roam, who can you trust ? Proc. *First Conference on Emerging Technologies and Applications in Communications*.

Orri, X. e Mas, J. M. (2001). Spki-xml certificate structure. Relatório técnico, Internet Engineering Task Force - Internet Draft., http://xml.coverpages.org/ni2001-12-18-a.html.

Oshima, M., Karjoth, G., e Ono, K. (1998). Aglets specification 1.1 draft. Relatório técnico, IBM Tókio.

Papaioannou, T. (2000). *On the Structuring of Distributed Systems: the Argument for Mobility*. Tese de Doutorado, Loughborough University.

Picco, G. P. (1998). *Understanding, Evaluating, Formalizing and Exploiting Code Mobility*. Tese de Doutorado, Politecnico di Torino, Italy.

Powell, D. (1988). An publication: The delta-4 projectconsortium. ISBN:2-907801-00-7. France.

Rabelo, R., Wangham, M., Schmidt, R., e Fraga, J. (2003). Trust building in the creation of virtual enterprises in mobile agent-based architectures. Proc. *4Th IFIP Working Conference on Virtual Enterprise*.

Rasmusson, L. e Jansson, S. (1996). Simulated social control for secure internet commerce. Em press, A., editor, *New Security Pardigms'96*.

Rasmusson, L., Rasmusson, A., e Jansson, S. (1997). Using agents to secure the internet marketplace – reactive security and social control. Proc. *Proc. 2nd Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*.

Riordan, J. e Schneier, B. (1998). Environmental key generation towards clueless agents. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*. Springer.

Rivest, R. L. (1997). *SEXP (S-expressions)*. The Internet Engineering Task Force-Internet Draft. http://theory.lcs.mit.edu/ rivest/sexp.html.

Roth, V. (1998). Secure recording of itineraries through cooperating agents. Proc. *ECOOP Workshop on Distributed Object Security and 4th Workshop on moble Object Systems: Secure Internet Mobile Computations*, pp. 147–154. INRIA.

Roth, V. (2001). On the robustness of some cryptographic protocols for mobile agent protection. Em Picco, G. P., editor, *Mobile Agents*, Vol. 2240 of *LNCS*, pp. 1–14. Springer.

Sakabe, Y. e Myaji, M. S. A. (2003). Java obfuscation with theorical basis for building secure mobile agents. Em Lioy, A. e Mazzocchi, D., editors, *7th IFIP-TC6 TC11 International Conference on Communications and Multimedia Security, CMS 2003*, Vol. 2828 of *LNCS*, pp. 89–103. Springer.

Saltzer, J. H. e Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, Vol. 63, No. 9, pp. 1278–1308.

Samar, V. e Lai, C. (1996). Making login services independent from authentication technologies. Proc. *In Proceedings of the SunSoft Developer's Conference*.

Sander, T. e Tschudin, C. (1998). Protecting mobile agens against malicious hosts. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*. Springer.

Sandhu, R. S. e Samarati, P. (1996). Authentication, access control, and audit. *ACM Computing Surveys*, Vol. 32, No. 9, pp. 40–48.

Santin, A., Fraga, J., Mello, E., e Siqueira, F. (2003). Extending the spki/sdsi model through federation webs. Proc. *Proc. of Seventh IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*.

Santin, A. (2004). *Teias de Federações: uma Abordagem baseada em Cadeias de Confiança para Autenticação, Autorização e Navegação em Sistemas de Larga Escala*. Tese de Doutorado, Universidade Federal de Santa Catarina.

Schmidt, R. (2003). Busca e seleção de parceiros para empresas virtuais: uma abordagem baseada em agentes móveis. Dissertação de Mestrado, Universidade Federal de Santa Catarina.

Schneider, F. B. (1997). Towards fault-tolerant and secure agentry. Em Mavronicolas, M. e Tsigas, P., editors, *11th Int. Workshop on Distributed Algorithms (WDAG '97)*, number 1320 in Lecture Notes in Computer Science, pp. 1–14. Springer-Verlag.

Schneier, B. (1996). *Applied Cryptography - protocols, algorithms and source code in C*. John Wiley and Sons, Inc.

Schneier, B. (2000). *Secret and Lies*. Wiley.

Stallings, W. (1998). *Cryptography and Network Security Principles and Practice*. Prentice Hall.

Steiner, J., Neuman, B., e Schiller, J. (1988). Kerberos: An authentication service for open network systems. Proc. *Winter USENIX Conference*, pp. 191–201.

Stoneburner, G., Goguen, A., e Feringa, A. (2001). Risk management guide for information technology systems. Relatório Técnico NIST Special Publication 800-30, National Institute of Standards and Technology.

Sundsted, T. (1998). An introduction to agents. Java World. http://www.javaworld.com.

Sun (2002). Java 2 sdk. v1.4 security documentation. http://www.java.sun.com/security/index.html.

Tardo, J. e Valente, L. (1996). Mobile agent security and telescript. Proc. *IEEE COMPCON'96*, pp. 58–63. IEEE Computer Society Press.

Thorn, T. (1997). Programming languages for mobile code. *ACM Computing Surveys*, Vol. 29, No. 3, .

Tripathi, A., Ahmed, T., e Karnik, N. (2001). Experiences and future challenges in mobile agent programming. *Microprocessors and Microsystems*, Vol. 25, No. 2, pp. 121–129.

Uto, N. e Dahab, R. (2001). Segurança de sistemas de agentes móveis. Proc. *3 Simpósio de Segurança em Informática*.

Uto, N. (2003). Segurança de sistemas de agentes móveis. Dissertação de Mestrado, UNICAMP.

Vigna, G. (1997). Protecting mobile agents tracing. Proc. *3rd ECOOP Workshop on Mobile Object Systems: Operating System support for Mobile Object Systems(MOS´97)*. Finlândia.

Vigna, G. (1998a). Cryptographic traces for mobile agents. Em Vigna, G., editor, *Mobile Agents and Security*, Vol. 1419 of *LNCS*, pp. 137–153. Springer.

Vigna, G., editor (1998b). *Mobile Agents and Security*, Vol. 1419 of *LNCS*. Springer.

Vitek, J., Serrano, M., e Thanos, D. (1997). Security communication in mobile object systems. Em Vitek, J. e Tschudin, C., editors, *Mobile Object Systems: Towards the Programmable Internet*, Vol. 1222 of *LNCS*. Springer.

Walsh, T., Paciorek, N., e Wong, D. (1998). Security and reliability in concordia. Em Society, I. C., editor, *31st Hawaii´s International Conference os System Sciences (HICSS´98)*.

Wangham, M. S., da Silva Fraga, J., e Obelheiro, R. R. (2003). A security scheme for mobile agent platforms in large-scale systems. Em Lioy, A. e Mazzocchi, D., editors, *Proc. of 7TH TC-6 TC-11 IFIP Conference on Communications and Multimedia Security*, Vol. 2828, pp. 104–116. Springer.

Wang, C., Knight, J., e Davidson, J. (2000). Software tamper resistance: Obstructing static analysis of programs. Relatório Técnico Tecnhical Report CS-2000-12, Department of Computer Science, University of Virginia.

Wang, C. (2000). *A Security Architecture for Survivability Mechanisms*. Tese de Doutorado, University of Virginia.

Westphall, C. M. (2000). *Um Esquema de Autorizaçao para Segurança em Sistemas Distribuídos de Larga Escala*. Tese de Doutorado, Universidade Federal de Santa Catarina.

Yee, B. (1997). A sanctuary for mobile agents. Proc. *Secure Internet Programming*, Vol. 1603 of *LNCS*, pp. 261–273. Springer.

Young, A. e Yung, M. (1997). Sliding encryption: A cryptographic tool for mobile agents. Proc. *4th International Workshop on Fast software Encryption, FSE'97*.