A ser publicado nos anais da EINE´99 (Escola de Informática da SBC – Edição Nordeste 1999). Salvador, Bahia. Novembro, 1999

Agentes Móveis

Flávio M. Assis Silva

LaSiD - Laboratório de Sistemas Distribuídos
CPD/UFBA
Av.Adhemar de Barros S/N
CEP 40170-110, Salvador - Bahia
email: fassis@ufba.br

Resumo

Um *agente móvel* (ou simplesmente *agente*) é um elemento de software auto-contido, responsável pela execução de uma atividade, que é capaz de *autonomamente migrar* através de uma rede. Um agente migra em um ambiente distribuído de uma *agência* a outra. Um *agência* representa um "lugar" lógico no sistema distribuído. Quando um agente migra, sua execução é suspendida na agência original, o agente é transportado (i.e., código, dados e estado de execução) para uma outra agência no ambiente distribuído, onde a execução é retomada.

Atualmente há uma série de projetos e produtos que implementam suporte a agentes móveis, como, por exemplo, Telescript, Mole, TACOMA, Aglets e Grasshopper. O conceito de agentes móveis vem sendo proposto para suportar diferentes tipos de aplicações, dentre elas: comércio eletrônico, gerenciamento de *workflows,* gerenciamento de redes, serviços de telecomunicação, recuperação de informações distribuídas e redes ativas.

Este texto apresenta: uma caracterização de agentes móveis; aspectos de sistemas e de ambientes de agentes móveis; uma argumentação sobre potenciais vantagens de agentes móveis; e um resumo dos principais sistemas de agentes móveis existentes.

1. Introdução

O termo *agente* tem sido usado em muitos contextos, denotando diferentes tipos de conceitos. Passou-se então a se discutir muito sobre as propriedades que um componente de software ou hardware deveria ter ou não para ser chamado de agente. Diferentes taxonomias para agentes foram propostas [Franklin, Graesser 96][Mendes, Loyolla et al. 96][Nwana 96], baseadas em propriedades tais como autonomia, inteligência, habilidade de comunicação, capacidade de movimento, entre outras. Em particular, as classificações tentaram integrar a noção de agentes desenvolvida pela comunidade de inteligência artificial com a noção de componentes de software móveis.

Neste texto agentes móveis são considerados simplesmente como componentes de software com capacidade de migrar de um nó a outro em um ambiente distribuído. Quando um agente se move, ele carrega consigo seu código, dados e estado corrente de execução. Outras propriedades que um agente pode ter, tais como inteligência ou o grau de habilidade de comunicação, são consideradas aqui ortogonais à definição de agentes móveis. Os termos *agente* e *agente móvel* serão usados indistintamente neste texto.

O desenvolvimento de uma infra-estrutura de suporte à execução de agentes móveis requer o desenvolvimento de soluções para um conjunto de problemas específicos que surgem com a migração. As soluções que vêm sendo propostas têm variado consideravelmente entre diferentes sistemas de agentes. O objetivo deste texto é, com isto: fornecer uma descrição de conceitos relacionados a agentes; discutir potenciais vantagens de agentes móveis; e fornecer uma descrição da funcionalidade que tem sido suportada ou que está em desenvolvimento em alguns dos principais projetos de pesquisa e produtos comerciais orientados a agentes móveis.

Este texto está estruturado da seguinte forma. A seção 2 apresenta agentes móveis como uma forma de paradigma de interação. Na seção 3, diferentes aspectos de sistemas de agentes móveis são apresentados. O conteúdo desta seção fornece um modelo de agentes móveis e de seu ambiente de execução. Na seção 4, benefícios potenciais de agentes móveis são discutidos. Na seção 5, alguns dos principais sistemas de agentes móveis são brevemente descritos.

2. Agentes Móveis e Paradigmas de Interação em Sistemas Distribuídos

A pesquisa em sistemas distribuídos desenvolveu-se na direção de se definir uma infra-estrutura de processamento que suporte um conjunto de *transparências* para as aplicações. Com transparências, aplicações podem ser desenvolvidas abstraindo-se de especificidades do ambiente, tais como a existência de uma rede de comunicação possivelmente complexa, heterogeneidade de software e hardware, falhas e concorrência. Transparências são definidas, por exemplo, na arquitetura OMG OMA (*Object Management Group - Object Management Architecture*) [OMG 92] e no padrão ODP (*Open Distributed Processing*) [ISO/IEC 94][ITU-T 95].

Um dos tipos de transparência é a *transparência de localização*. Em um ambiente distribuído que fornece transparência de localização, dois componentes podem-se comunicar sem que tenham conhecimento do local no ambiente onde o componente par está.

Em sistemas distribuídos tradicionais, componentes clientes e servidores interagem uns com os outros através de alguma forma de RPC (*Remote Procedure Call*, chamada remota de procedimento) ou de seu correspondente na orientação a objetos RMI (*Remote Method Invocation*, invocação remota de método). Em tais ambientes, cada componente permanece em um único local durante todo seu ciclo de vida. Considere dois componentes A e B que se interagem, localizados respectivamente nos nós S_A e S_B. O componente B oferece um conjunto de serviços. Quando o componente A quer usar um serviço oferecido por B, o componente A executa uma chamada remota de procedimento/invocação remota de método através da rede para invocar a execução do serviço no nó S_B. Se o pedido retorna resultados, estes serão eventualmente transmitidos a A (veja figura 1). O pedido pode ser feito ou *sincronamente*, isto é, com o componente A ficando bloqueado até que os resultados retornem, ou *assincronamente*, quando o componente A pode continuar a executar sua atividade logo depois de ter feito a invocação e mais tarde obter os resultados.

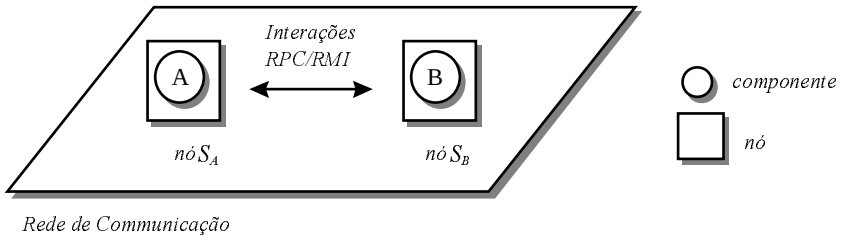

Figura 1: Comunicação através de RPC/RMI

Agentes móveis suportam uma forma diferente de paradigma de interação. Considere um componente (agente) A inicialmente localizado e executando em S_A. Em um certo ponto da execução, A quer acessar certos recursos no nó S_B. Para acessar estes recursos, o componente A *se move* de S_A para S_B, carregando consigo os seus dados e informação sobre o seu estado de execução corrente (veja figura 2a). A execução da lógica de A é interrompida no nó S_A e é retomada em S_B, depois do movimento. Depois de acessar os recursos desejados em S_B, o componente A pode-se mover para outro local no ambiente distribuído, por exemplo, para o nó S_C (veja Figura 2b), carregando novamente consigo o estado corrente de seus dados e informação sobre o seu estado corrente de execução.

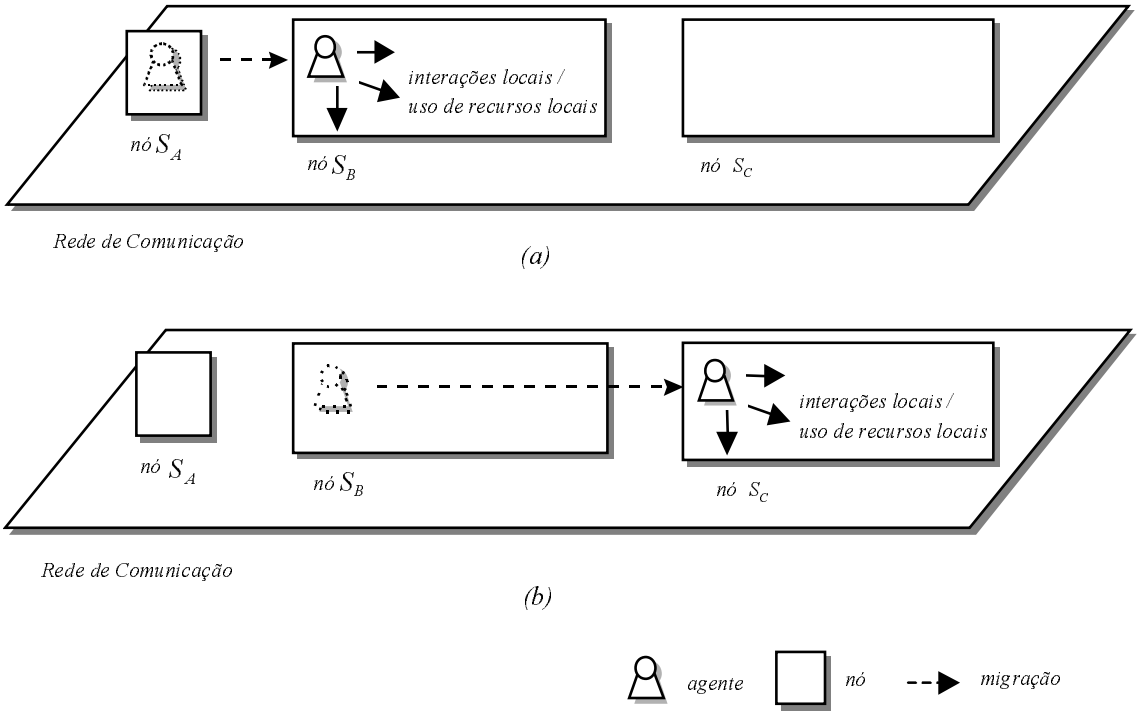

Figura 2 : Agentes móveis

Agentes móveis podem-se mover para um nó para acessar recursos *naquele* nó ou *próximo* dele. Por exemplo, um agente móvel pode migrar para um nó para acessar recursos espalhados através da LAN (*Local Area Network*) onde o nó está localizado.

A migração de agentes móveis pode ser classificada em *forte* ou *fraca* [Fuggetta, Picco, Vigna 98]. Na migração forte, a informação sobre o ponto na execução onde a migração foi requerida é automaticamente obtida pelo suporte de execução (*runtime*

support) e é transmitida juntamente com o agente. O agente pode retomar a execução depois da migração *exatamente* do ponto onde parou. Na migração fraca, nenhuma informação deste tipo é *automaticamente* obtida pelo suporte de execução. O sistema suporta apenas ou que o agente reinicie no nó remoto de um ponto de início convencionado (o mesmo método é chamado pelo sistema a cada vez que um agente retoma execução em um nó) ou que o agente especifique qual de seus métodos deve ser executado depois da migração. Na migração fraca o contexto necessário para se resumir a execução da atividade do agente depois da migração tem que ser definido utilizando-se os dados do agente.

A execução de atividades com agentes baseia-se em uma noção de *localidade* no ambiente distribuído. Há no ambiente informação de *onde* certos recursos estão. A transparência de localização não é, portanto, completamente suportada. A noção de localidade, no entanto, é usada para se atingir certos objetivos para as aplicações. Fornecer transparência de localização em ambientes distribuídos é útil para se esconder de aplicações a complexidade do sistema, mas impede que aplicações se beneficiem de vantagens que alguma forma de conhecimento de localidade pode trazer. Estas vantagens serão discutidas na seção 4.

O conceito de agentes móveis se diferencia do conceito de migração de processos em sistemas operacionais distribuídos principalmente pelo fato de o *agente móvel por si* decidir quando deve mover. Com agentes móveis uma aplicação pode explorar ela mesma a possibilidade de mover código para atingir objetivos próprios. No caso de migração de processos, o movimento de processos é decidido pelo *ambiente de suporte* e é feito para se atingir um certo objetivo *a nível do sistema*, por exemplo, balanceamento de carga.

Agentes móveis representam uma forma específica de *mobilidade de código*. Além de agentes móveis, outras duas formas de mobilidade de código podem ser citadas [Fuggetta, Picco, Vigna 98]:

- *avaliação remota* : na avaliação remota, um trecho de código é enviado a um local (nó) remoto, possivelmente com parâmetros iniciais, para executar naquele local. O código é executado a partir do início. Depois de ser iniciado, o trecho de código executará naquele local remoto até seu término (veja figura 3);

- *código sob demanda* : no código sob demanda, um componente de software executando em um local requisita o carregamento naquele local de um trecho de código existente em um local remoto (veja figura 4). No código sob demanda, o trecho de código será executado também até seu término no mesmo local.

Enquanto na avaliação remota e no código sob demanda somente trechos de código são transferidos (possivelmente com alguns parâmetros iniciais), no caso de agentes móveis não somente código e dados mas também o estado de execução é transferido, isto é, uma unidade de execução completa. Com agentes móveis o *fluxo de controle* de uma atividade se move através da rede.

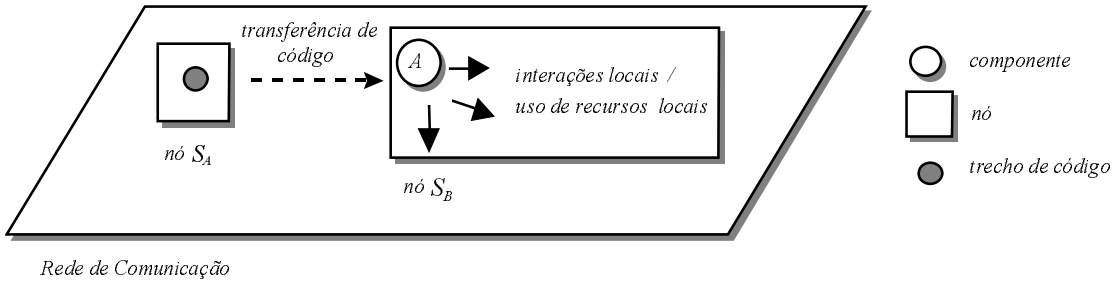

Figura 3 : Avaliação remota

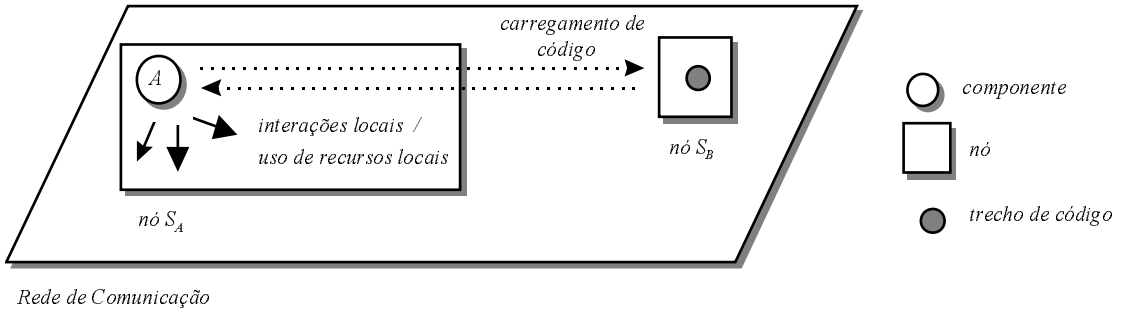

Figura 4 : Código sob demanda

3. Agentes Móveis e seu Ambiente

3.1. Agências e o DAE

O ambiente onde agentes executam é chamado neste texto de DAE (*Distributed Agent Environment* - Ambiente de Agente Distribuído) (veja figura 5). A descrição do DAE feita a seguir é baseada na proposta de padrão MASIF [OMG 98], que descreve um ambiente de agente genérico.

No DAE um agente se move de uma *agência*[1] a outra. Uma agência representa o suporte de execução para todo o ciclo de vida dos agentes (criação, execução, término, transferência de agentes, etc.). Uma agência é localizada em um nó. Um nó pode, no entanto, conter mais de uma agência (por exemplo, uma agência Grasshopper e uma agência Voyager. Veja seção 5).

Uma agência é estruturada em *lugares*. Um lugar representa um contexto no qual um agente pode executar. Diferentes lugares podem fornecer diferentes contextos para agentes que chegam na agência, por exemplo, para suportar diferentes formas de controle de acesso ou políticas de contabilização de recursos (*accounting*).

[1] No MASIF, uma agência é chamada de um *sistema de agente.*

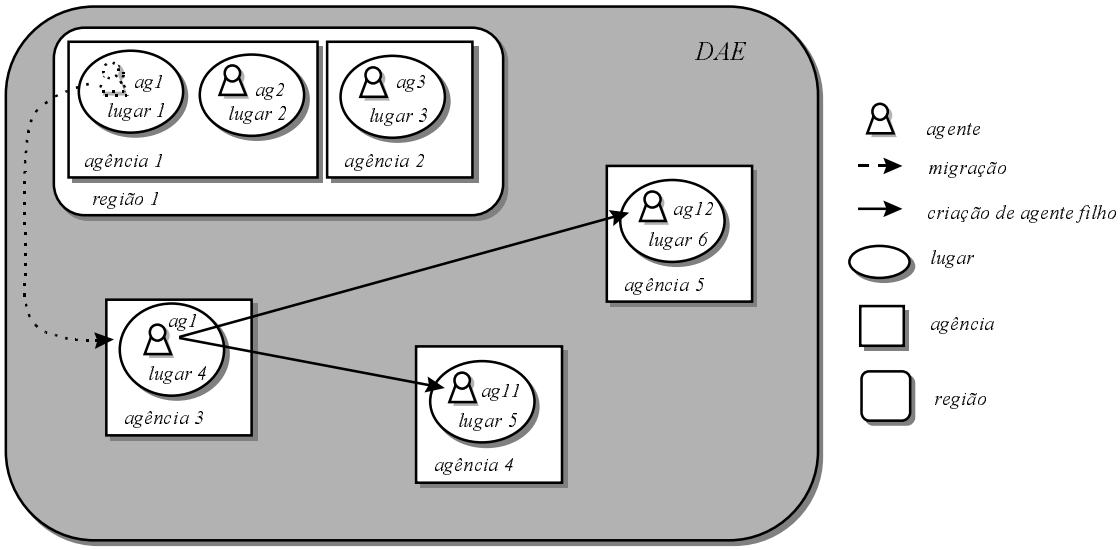

Figura 5 : O DAE (Ambiente de Agente Distribuído)

Agências podem ser agrupadas para formar uma *região*. Todas as agências de uma região pertencem à mesma organização. Uma região representa uma área lógica maior para onde um agente pode ir. Uma região tem um subconjunto de suas agências designadas como os *pontos de entrada* da região. Quando um agente chega em um ponto de entrada de uma região, ele pode ser efetivamente posto para executar em qualquer agência da região.

Durante sua execução, um agente móvel pode criar outros agentes móveis. Neste texto, os agentes móveis criados serão chamados de *agentes filhos* do agente móvel que os criou, o *agent pai*. Um agente filho pode ser criado ou na mesma agência onde o agente pai está ou em alguma outra agência. Um agente filho executa e migra autonomamente em relação a seu agente pai. A figura 5 mostra dois agentes filhos, *ag11* e *ag12*, criados pelo agente *ag1* enquanto ele está rodando na *agência 3*. O agente filho *ag11* foi criado na *agência 4*, enquanto *ag12* foi criado na *agência 5*. Usando-se agentes filhos, partes de uma atividade baseada em agentes podem ser executadas assincronamente em relação a outras partes.

Cada agente, agência, lugar e região tem um identificador que o(a) identifica de maneira não ambígua em todo o ambiente. Um agente móvel mantém o mesmo identificador enquanto se move de agência para agência. Uma agência tem também um *endereço de agência*[2], que determina sua localização no DAE. Um agente tem a cada momento um *lugar corrente do agente*, que é o conjunto do endereço da agência mais a identificação do lugar onde o agente está no momento.

A expressão *sistema de agente*[3] será usada aqui para se referir a uma tecnologia que fornece uma infra-estrutura para suportar a execução de agentes móveis. Exemplos de sistemas de agentes são Grasshopper, Voyager, IBM Aglets, Mole, etc.

---

[2] Chamado de um *endereço de sistema de agente* no MASIF

[3] Chamado de um *tipo de sistema de agente* no MASIF

3.2. Estrutura de Agentes

Estruturalmente, um agente tem *dados internos, código* e *meta-dados* (ou *atributos* do agente).

Os dados internos do agente representam a informação que o agente processa durante sua execução. Tais dados representam dados colecionados através de interações (com aplicações, seu usuário ou outros agentes), dados criados pelo agente e valores de parâmetros iniciais recebidos pelo agente no momento de sua criação. Enquanto o agente se move através do ambiente, pode ser necessário que parte de seus dados seja armazenada de maneira persistente.

O código do agente representa o conjunto de instruções que determina a lógica do agente.

Os meta-dados de um agente representam informações sobre o agente e sua execução. Por exemplo, o identificador do agente, seu *status* operacional (ativo, suspenso, etc.) e a organização à qual o agente pertence.

A qualquer instante durante sua execução, um agente tem um *estado de execução* e um *estado de seus dados*. O estado de execução representa a informação que descreve o ponto exato do controle de fluxo onde a execução de um agente está em um certo instante. O estado dos dados do agente representa o valor de todos os dados internos em um instante considerado. Adicionalmente, um *estado operacional do agente* pode ser definido para indicar o seu *status* operacional.

3.3. Linguagem de Programação de Agentes

O código do agente é programado em uma *linguagem de programação de agentes*. Diferentes linguagens estão sendo usadas ou sendo propostas para se programar agentes móveis, entre elas Java [Gosling, McGilton 96], Obliq [Cardelli 95], Tcl/Tk [Ousterhout 95] e Facile [Knabe 95]. Sistemas de agentes estão suportando (ou planejam suportar) linguagens de programação de agentes imperativas, declarativas e funcionais.

Um requisito fundamental para uma linguagem de programação de agentes é o suporte a portabilidade, devido à natureza heterogênea de ambientes distribuídos para os quais se prevê a utilização de agentes. Para suportar heterogeneidade, uma linguagem de programação de agentes é preferível se ela é diretamente interpretada ou compilada para um código intermediário portável que pode ser interpretado em cada nó. Trechos de código escritos em linguagens que são compiladas para linguagens de máquina específicas de arquiteturas têm que ser transportados como código fonte entre sistemas heterogêneos. O código então tem que ser recompilado no sistema destino antes que o agente possa retomar a execução. Linguagens baseadas em alguma forma de interpretação suportam melhor a portabilidade do código dos agentes e evitam a recompilação no sistema para onde um agente migrou. Além disto, mecanismos de segurança podem ser melhor aplicados com linguagens interpretadas do que compiladas. Suporte a linguagens compiladas, no entanto, não está excluído de sistemas de agentes. Tal suporte é considerado, por exemplo, no projeto TACOMA [Johansen, Schneider, van Renesse 98].

É objetivo de alguns projetos de agentes móveis suportar mais do que uma linguagem de programação de agentes. Diferentes linguagens podem fornecer ao

programador diferentes capacidades de programação. Uma agência que suporta agentes programados em diferentes linguagens tem que ter um interpretador ou compilador para cada linguagem suportada.

Em Telescript uma linguagem de programação de agentes específica foi desenvolvida, chamada *linguagem Telescript* [White 94b]. Esta linguagem era persistente e requeria hardware especial. Projetos em universidades foram depois iniciados que passaram a usar linguagens mais geralmente disponíveis e que poderiam ser suportadas em potencialmente qualquer tipo de sistema. Nos sistemas TACOMA e AgentTcl, por exemplo, Tcl/Tk é usada [Johansen, van Renesse, Schneider 95a][Gray 95]. Para se fornecer migração forte, um interpretador específico para esta linguagem foi desenvolvido no sistema AgentTcl [Gray, Cybenko et al. 96].

Com o advento de Java [Gosling, McGilton 96], Java tornou-se a linguagem de programação de agentes móveis mais usada. Java tem uma série de propriedades apropriadas para uma linguagem de programação de agentes. Programas em Java são compilados para um formato objeto independente de arquitetura chamado Java Byte-code, o que torna os programas altamente portáveis. Java é orientada-a-objetos, robusta, suporta *threads*, permite resolução dinâmica de referências de código (*dynamic binding*) e fornece um mecanismo de segurança embutido. Além disto, Java tem uma forte aceitação pelo mercado como uma linguagem de propósito geral e inúmeros produtos vêm sendo desenvolvidos para Java, tais como ferramentas de persistência, pacotes de segurança, bibliotecas de propósito geral, etc.

A maioria dos principais sistemas de agentes suporta Java como a linguagem de programação de agentes ou pelo menos uma das linguagens. Com exceção do trabalho descrito em [Fünfrocken 98], todos os outros principais sistemas baseados em Java suportam migração fraca. Em [Fünfrocken 98] o autor descreve um mecanismo para obter e recuperar o estado de programas Java de maneira transparente. O mecanismo trabalha a nível da linguagem, usando um pré-processador que adiciona código ao programa para controlar o armazenamento e recuperação de informação relevante. Este mecanismo fornece uma forma de migração forte. Migração fraca, entretanto, não parece fornecer uma barreira para a programação de agentes, como era considerado no início da pesquisa em agentes móveis. Parece até mesmo que migração forte pode não ser necessária [Baumann, Hohl et al. 97].

3.4. Comunicação de Agentes

Sistemas de agentes móveis variam bastante nas formas de comunicação suportadas. As diversas formas de comunicação podem ser divididas em dois grupos: *comunicação direta*, onde os componentes comunicantes explicitamente se identificam; e *comunicação anônima*, onde o emissor não conhece a identidade dos recebedores. Comunicação direta pode envolver somente dois componentes (agentes, objetos) ou um emissor e um conjunto de recebedores (*multicast*). Formas de comunicação anônima sendo suportadas por sistemas de agentes são *eventos, espaços (spaces)* e *espaços de tuplas (tuple spaces*). Nenhum dos sistemas de agentes suporta todas as formas de comunicação descritas aqui.

O conceito de *meet* (encontro) [White 94b][Johansen, van Renesse, Schneider 95a] é uma forma de comunicação direta. O conceito de *meet* suporta que dois agentes comunicantes “se encontrem” em uma certa agência para então interagirem. Depois de

se encontrarem, os agentes podem ou invocar métodos de suas interfaces ou simplesmente trocar dados. Em sistemas de agentes móveis é comumente suportado agora, no entanto, que agentes móveis possam comunicar ou estando na mesma agência ou estando em agências diferentes.

Sistemas de agentes suportam formas de comunicação direta síncrona e assíncrona. Mensagens assíncronas podem ser ou *oneway*, em que nenhum resultado é esperado, ou *future*, em que resultados são recebidos pelo agente emissor algum tempo depois. Mensagens dinâmicas são também suportadas, onde a mensagem e seus parâmetros são definidos no momento em que a mensagem será enviada. Antes de se enviar mensagens, o estabelecimento de uma *sessão* pode ser necessário.

Uma forma de se enviar mensagens em *multicast* onde o emissor sabe a identidade dos recebedores é descrita em [Leichsenring 98] (veja seção 5.8).

A identificação de agentes para o estabelecimento de comunicação é comumente baseada em um identificador de agente. Em [Baumann, Hohl et al. 97] uma forma flexível de se identificar agentes móveis baseada em identificadores definidos pela aplicação (chamados *badges*) é descrita. A forma como *badges* são usados está descrita na seção 5, na parte que descreve o sistema Mole.

Para se enviar uma mensagem ou invocar o método de um agente pode ser necessário que sejam especificados os lugares onde os agentes comunicantes estão. Alguns sistemas suportam que o envio de mensagens seja feito independentemente de onde o agente destino está e até mesmo que os agentes comunicantes possam-se mover durante a comunicação [ObjectSpace 97][IKV++ 98].

A comunicação através de eventos requer que consumidores e (possivelmente também) produtores de eventos se registrem em um coordenador de eventos, indicando, respectivamente, os tipos de eventos pelos quais estão interessados ou que irão gerar. Uma forma flexível de se gerenciar eventos é proposta em [Baumann, Hohl et al. 97] onde *objetos de sincronização* recebem eventos de entrada e geram eventos de saída. Os eventos de saída dependem de eventos de entrada, regras internas, informação de estado e intervalos de *timeout* (veja mais detalhes na descrição do sistema Mole na seção 5).

O conceito de *espaço* (*space*) [ObjectSpace 97] suporta uma forma de comunicação do tipo *publish/subscribe*. Grupos de objetos são agrupados em chamados *sub-espaços*. Os sub-espaços são então unidos para formar um grupo maior, chamado de *espaço*. Quando uma mensagem/evento é enviada(o) em um sub-espaço, ela(e) é copiada(o) e enviada(o) para cada um dos sub-espaços vizinhos antes que seja entregue a todos os objetos no sub-espaço local. Através de um mecanismo específico em cada sub-espaço, nenhuma mensagem/evento é acidentalmente processada(o) mais do que uma vez, independentemente de como os sub-espaços estão unidos. Quando um grande número de objetos está envolvido, espaços fornecem melhor extensibilidade do que um método de disseminação baseado em um servidor centralizado. Uma forma de tolerância a falha é também suportada, já que se uma mensagem/evento chega em um sub-espaço ela(e) é propagada(o) para todos os sub-espaços do mesmo espaço. Um objeto (agente) que quer enviar uma mensagem para o grupo *publica* (*publish*) a mensagem. Objetos (agentes) que querem receber mensagens, *subscrevem* (*subscribe*) para mensagens do tipo desejado.

Espaços de tuplas representam um espaço de informação onde agentes podem escrever e ler informação estruturada na forma de *itens* (*k, a, v*) [Lingnau, Drobnik 96]. Em um item, *k* representa uma chave que identifica o item, *a* representa informação de controle de acesso e *v* representa o valor do item. Itens são armazenados e recuperados do espaço atomicamente. Informação é recuperada do espaço de maneira associativa, utilizando-se algum mecanismo de casamento de padrões. Espaços de tuplas podem ser feitos *reativos*, onde ações determinadas podem ser executadas quando acessos específicos ao espaço de tuplas forem feitos [Cabri, Leonardi, Zambonelli 98].

O suporte à troca de informação ao nível semântico, tal como suporte para uma linguagem como KQML/KIF [Finin, Labrou, Mayfield 95], é considerado aqui ortogonal a sistemas de agentes móveis. Por isso tal suporte não é discutido neste texto.

3.5. Persistência de Agentes/Tolerância a Falhas

Tolerância a falhas de execuções baseadas em agentes requer pelo menos: alguma forma de se suportar persistência de estado de agentes; um mecanismo para se reativar agentes depois de uma falha; um mecanismo para recuperar o estado da atividade do agente depois de uma falha; e uma forma de se transportar agentes entre agências de maneira confiável. A execução de atividades baseadas em agentes móveis deveria ser capaz de tolerar também *falhas de longa duração* de agências. Se a agência onde o agente está executando falha, a execução do agente fica bloqueada enquanto a agência permanecer falhada. É desejado que a execução do agente possa ser recuperada em uma outra agência, de onde ela pode ser continuada.

Alguns sistemas de agente fornecem suporte para que o estado de agentes móveis possa ser armazenado e lido do disco em agências. Persistência de agentes é importante não somente para se recuperar a execução de agentes no caso de falhas, mas também como um meio de se gerenciar o uso de memória em agências. Agentes que não estão ativos executando uma tarefa (por exemplo, simplesmente aguardando por alguma mensagem) podem ser armazenados em disco e reativados mais tarde. Suporte para persistência tem estado presente em sistemas de agentes como um serviço que permite que agentes requisitem que seu estado seja armazenado ou lido. Este serviço tem sido comumente oferecido utilizando-se serialização de objetos [Oshima, Karjoth, Ono 98][ObjectSpace 97][IKV++ 98]. Mecanismos para se carregar agentes automaticamente quando uma agência inicia sua execução são também suportados.

Os enfoques descritos em [Oshima, Karjoth, Ono 98][ObjectSpace 97][IKV++ 98] armazenam somente o estado dos dados de agentes. O enfoque em [Fünfrocken 98], descrito anteriormente, pode ser usado para tornar o estado da execução também persistente para agentes programados em Java. Enfoques para tornar a linguagem Java persistente também estão sendo propostos [Jordan, Atkinson 98][Howell 98]. Como tais enfoques podem ser aplicados a sistemas de agentes móveis, no entanto, não tem sido ainda muito explorado na literatura.

Enfoques para tolerância a falhas de agentes móveis são propostos em [Schneider 97][Rothermel, Straßer 98][Assis Silva, Popescu-Zeletin 98][Assis Silva 99][Vogler, Kunkelmann, Moschgath 97][Johansen, Marzullo et al. 98]. Com exceção de [Vogler, Kunkelmann, Moschgath 97], os demais enfoques recuperam uma atividade baseada em agentes de falhas de agências de curta e longa duração. Tolerância a falhas de agentes

não será tratada aqui em mais detalhes. Em [Assis Silva 99] este tema é abordado em detalhes.

3.6. Segurança

Suporte a segurança em sistemas de agentes móveis envolve segurança da agência e segurança de agentes [Busse, Covaci, Nietfeld 98][Baumann, Hohl et al. 97].

Segurança da agência envolve proteger os recursos da agência contra acessos maliciosos ou abusivos de agentes ou outros clientes através de interfaces internas e externas da agência e proteger a informação transmitida durante interações através da rede. Tais garantias de segurança têm que existir também para qualquer serviço de suporte distribuído, como, por exemplo, serviços de nomes.

Segurança de agentes envolve proteger um agente contra acessos maliciosos de outros agentes, do sistema onde o agente está ou de outros clientes de suas interfaces. A informação sobre um agente durante um transporte e a informação trocada durante interações com aquele agente têm que ser também protegidas contra ataques.

A proteção durante comunicações através da rede e acessos a interfaces externas de uma agência envolvem fornecer mecanismos para: autenticar os componentes pares comunicantes, de forma que cada componente certifique que está realmente comunicando com quem ele supõe que seja; autorizar e fazer controle de acesso sobre os serviços fornecidos nas interfaces externas; garantir a confidência de mensagens transmitidas, de forma que a informação permaneça conhecida somente pelos componentes comunicantes; garantir a integridade da informação transmitida; e permitir auditoria, isto é, processar um registro da história de acessos feitos.

A proteção de serviços e recursos em uma agência, quando acessados através das interfaces internas da agência, requer mecanismos para se autenticar o agente que acessa o recurso; executar autorização/controle de acesso sobre os serviços e recursos acessados; e permitir auditoria.

Os problemas de autenticação de componentes comunicantes através da rede, confidência e integridade da informação transmitida têm sido resolvidos com o uso de protocolos de segurança tais como SSL (*Secure Socket Layer*) [Freier, Karlton, Kocher 96].

Suporte a controle de acesso, autenticação de um agente acessando recursos em uma agência e auditoria estão sendo fornecidos com o uso de um *gerenciador de segurança* fornecido pela linguagem usada para programar sistemas de agentes. Java fornece um mecanismos destes [Gong 98]. Com este mecanismo, permissões podem ser atribuídas considerando-se cada método potencialmente perigoso que pode ser chamado. Quando um tal método é invocado, o gerenciador de segurança checa a permissão de acordo com a política de segurança que estiver sendo aplicada e permite ou nega o acesso. A política de controle pode ser estendida para controlar a quantidade em que cada recurso pode ser usado.

A proteção de agentes contra acessos maliciosos em suas interfaces pode também ser feita com o uso do gerenciador de segurança [Busse, Covaci, Nietfeld 98]. Em uma linguagem tipada tal como Java, acessos a um objeto têm que ser feitos com uma referência ao objeto e somente os métodos da interface do objeto podem ser chamados.

Pode ser então feito que referências ao agente somente sejam obtidas por um cliente se a política de acesso implementada pelo gerenciador de segurança o permitir.

Dois aspectos de segurança de agentes, no entanto, parecem ser muito difíceis de serem resolvidos ou parecem ser insolúveis de uma maneira geral: proteger agentes contra o sistema onde o agente está rodando e suportar segurança quando agentes atravessam múltiplas agências (*multihopping*).

A dificuldade do primeiro problema aparece do fato que a agência tem que ter bastante controle sobre o agente e seu conteúdo para poder executá-lo. Possíveis ataques de uma agência sobre o agente são, por exemplo, a captura de informações privadas do agente. Em um cenário de comércio eletrônico, uma agência vendendo algum produto para um agente pode investigar a política que o agente usa para decidir sobre uma compra e usar esta informação para maximizar o lucro que ele pode obter com a transação [Busse, Covaci, Nietfeld 98].

Alguns enfoques têm sido propostos para tratar deste problema [Baumann, Hohl et al. 97]. Dois deles são: confiança/reputação, onde agentes migram apenas para nós que são confiáveis ou que têm boa reputação; e utilizando-se técnicas de ofuscação de código juntamente com um mecanismo de *timeout*. No segundo enfoque, as ações de um agente se tornam inválidas depois de um certo período de tempo. A idéia é que este período de tempo protege o agente por um certo intervalo, já que o componente que o ataca precisa de tempo para analisar o código ofuscado do agente. Estes e outros enfoques para o problema são, no entanto, temas de trabalho em andamento.

O problema que surge no caso de *multihopping* é que uma agência deveria confiar em um agente dependendo de como esta agência confia nas outras agências onde o agente executou anteriormente, já que cada agência pode ter modificado o agente. Uma solução baseada na assinatura digital das modificações feitas no agente, feita por cada agência visitada pelo agente, parece ser difícil de ser realizada, devido à quantidade de computação que envolveria [Busse, Covaci, Nietfeld 98]. Parece que uma solução para segurança no caso de *multihopping* deverá ser baseada em confiança e reputação dos sistemas onde agentes móveis executam. Uma agência teria confiança em um agente que chega somente se ele executou exclusivamente em agências em que a agência simplesmente confia.

3.7. Contabilização (*Accounting*)

Contabilização está relacionada à cobrança ou à limitação dos recursos usados por um agente enquanto ele roda em uma agência. Exemplos de recursos que podem ser gerenciados são: tempo de CPU, comunicação com componentes em redes remotas, número de agentes filhos criados e tempo total que o agente fica em um lugar. O gerenciamento de CPU, por exemplo, pode ser calculado contando-se os *time slices* dedicados aos *threads* de um agente. O tempo total gasto em um lugar é computado registrando-se o tempo quando o agente chegou no lugar. O gerenciamento de recursos de comunicação e da quantidade de agentes filhos pode ser feito pela agência já que o uso destes recursos é feito utilizando os recursos da agência [Baumann, Hohl et al. 97].

Pode-se também permitir ao programador de um agente conceder um tempo de vida máximo, um tamanho máximo e uma quantidade máxima de uso de recursos a seus agentes [White 94b]. Com isto pode-se limitar a quantidade de recursos usados por um agente durante sua execução. Desta maneira evita-se que agentes com erro de

programação inadvertidamente comecem a consumir recursos no ambiente sem controle.

Pelos conhecimentos do autor deste texto, não há ainda na literatura de agentes uma descrição mais completa de como contabilização pode ser feita em um ambiente de agentes.

3.8. Autonomia em Sistemas de Agentes

O conceito de *autonomia* é comumente associado a agentes. A noção de autonomia aplicada a sistemas de agentes é, no entanto, sujeita a diferentes interpretações. Neste texto, duas formas de se entender autonomia no contexto de sistemas de agentes são apresentadas [Assis Silva 99].

Primeiramente, considera-se que autonomia representa simplesmente o mesmo que *a habilidade de um agente executar sua tarefa de maneira assíncrona*. Um agente é emitido de um lugar em um ambiente distribuído e pode migrar neste ambiente sem nenhuma conexão com o sistema original. Neste sentido, autonomia é entendida como o fato de o agente *ser capaz* de executar sua tarefa *independentemente da disponibilidade ou controle do sistema original*. Por ser um programa, o código de um agente representa uma certa lógica, o que o habilita a executar uma determinada tarefa (seu objetivo) e reagir a diferentes condições do ambiente.

A segunda forma na qual autonomia é entendida é como uma representação do grau de controle ou influência que uma entidade (por exemplo, uma organização ou um agente) *pode ter* sobre aspectos (como comportamente ou estrutura) de uma outra entidade (tal como um provedor de um serviço ou uma agência). Se uma entidade, digamos *entidade_a*, é completamente autônoma em relação a uma outra entidade, digamos *entidade_b*, em relação a algum aspecto, *entidade_b* não tem nenhuma forma de controle ou influência sobre a *entidade_a* em relação àquele aspecto.

Considerando-se o conceito de agências, se alguma agência é autônoma em relação a seu comportamento, por exemplo, esta agência pode, sem qualquer influência de qualquer outra entidade (agências ou organizações, etc.), decidir exatamente como ela deve se comportar. A agência pode, então, por exemplo, autonomamente definir políticas para controlar quais agentes ela irá aceitar, quais preços serão aplicados para o uso de recursos, etc.

Como um outro exemplo, aplicando-se a noção de autonomia a organizações que emitem agentes para executar no DAE, estas organizações deveriam ter tanto controle quanto possível sobre a execução de seus agentes. A execução de atividades baseadas em agentes de uma organização não deveria ser dependente de políticas aplicadas por outras agências.

Suportar a autonomia de organizações e de agências desta maneira pode ser entendido como suportar a chamada *M-autonomia* de organizações, como definida em [Veijalainen, Eliassen, Holtkamp 92]. Aspectos de autonomia são importante no contexto de agentes móveis executando em um ambiente aberto (*open environment*), já que os agentes se movem para executar atividades em agências autônomas.

4. Benefícios Potenciais de Agentes Móveis

RPC/RMI, agentes móveis, avaliação remota e código sob demanda não devem ser considerados cada um como um paradigma que tem vantagens absolutas sobre os outros em todos os casos, mas eles devem ser considerados como paradigmas complementares. Ao se projetar uma certa aplicação, cada um deles deve ser considerado e comparado, de tal forma que a melhor alternativa possa ser escolhida em cada caso específico.

Agentes móveis podem ser usados para se explorar os seguintes benefícios potenciais:

- *descentralização de processamento*: agentes móveis possibilitam uma forma descentralizada de se controlar a execução de uma atividade distribuída. Em sistemas baseados em alguma forma de controle centralizado, o conjunto de nós de onde o controle é feito tem que suportar o processamento de todas as atividades sendo executadas no ambiente distribuído. A distribuição do controle das atividades é um aspecto importante a ser considerado quando se projetam sistemas que devem ser altamente extensíveis ou que devem rodar em ambientes de sistemas altamente autônomos. O uso de sistemas baseados em agentes é proposto como uma forma de se evitar limitações de sistemas centralizados, por exemplo, em [Magedanz, Eckardt 96][Magedanz, Popescu-Zeletin 96][Chess, Harrison, Kershenbaum 94][Alonso, Kamath et al. 94][Cai, Gloor, Nog 96][Merz, Liberman et al. 96].

  Além disto, se a execução de uma certa atividade envolve acessar somente recursos em nós remotos, não há necessidade de a execução ser dependente de nenhum modo do nó original onde a atividade iniciou. Por exemplo, depois de se executar parte da atividade em um nó, não há conceitualmente nenhuma razão para o fluxo de controle da execução ter que retornar ao nó original para simplesmente determinar como a atividade deve continuar. Falhas do nó original ou da comunicação com ele não deveriam afetar a execução da atividade, já que ela não depende de recursos lá. Agentes móveis permitem que a execução de uma atividade seja mais dependente somente dos nós onde a atividade *realmente* tem que ser executada;

- *bom suporte para o projeto de aplicações que interagem com usuários humanos em um ambiente distribuído*: em aplicações que envolvem interações com usuários humanos, o tempo de resposta *durante a interação* é importante. Se a interação é iniciada *pelo usuário humano*, código sob demanda é bastante conveniente para se carregar o código que controla a interação com o usuário em sua máquina. Isto é evidenciado pelo uso freqüente de *applets* em aplicações existentes. Agentes móveis podem ser usados *por uma aplicação* para iniciar (acionar) uma interação com um usuário humano *durante a execução* de uma aplicação distribuída e com isto obter as mesmas vantagens que código sob demanda;

- *melhor uso de recursos de comunicação*: um agente pode-se mover para uma agência para acessar recursos naquela agência com uma *qualidade de serviço* otimizada, evitando-se os efeitos causados por conexões de rede de qualidade inferior. Agentes móveis podem processar informação em um nó remoto e somente retornar o resultado final. Isto pode diminuir a quantidade de dados

que devem ser transferidos através da rede. Em [Gray 95] os autores descrevem que aplicações de consultas a bancos de dados de relatórios médicos e de gravuras de projeto gráficos representam dois casos onde agentes móveis podem fornecer uma melhora no uso de recursos de comunicação. Em tais sistemas, consultas podem variar bastante e não se pode esperar que os nós com os bancos de dados tenham todas as possíveis consultas como unidades atômicas. Um outro exemplo onde o uso de agentes móveis contribui para se evitar os efeitos de canais de comunicação através de uma rede é o uso de agentes móveis para executar monitoração local de componentes em cenários de gerenciamento de redes;

- *melhor uso de recursos computacionais*: agentes móveis possibilitam que a execução de uma atividade se mova para sistemas com propriedades especiais em momentos específicos durante a execução. O movimento é feito para se atingir objetivos específicos de uma aplicação. Por exemplo, se a execução de uma parte de uma atividade envolve um cálculo pesado, o agente móvel executando a atividade pode-se mover para um nó com suporte especial para processamento matemático. Se mais tarde um outro tipo de recurso especial é necessário, o agente móvel pode mover a execução para um outro sistema que tem aquele recurso. Utilizando-se mais de um agente móvel, uma forma paralela de processamento de uma certa computação pesada pode ser atingida, distribuindo cada parte de uma tarefa global para ser executada em computadores apropriados;

- *adaptação ao padrão assíncrono de execução de sistemas*: agentes móveis executam assincronamente, sendo enviados de uma maneira *armazena-e-passa-adiante* (*store-and-forward*). Se uma certa atividade deve ser executada em um sistema e este sistema não está sempre ativo, não está pronto ainda para executar a atividade ou não deseja ainda executar a atividade, um agente móvel pode ser enviado para aquele sistema e a atividade será executada quando possível ou quando aquele sistema decidir que é o momento mais conveniente para executá-la. Agentes móveis fornecem uma forma de se transferir para aquele sistema as atividades a serem executadas sem o sistema original ter que periodicamente testar se o sistema destino está ativo ou pronto. De uma maneira natural, quando o sistema remoto estiver pronto para receber novas tarefas (agentes), ele simplesmente carregará o conjunto de agentes móveis a serem executados lá;

- *adaptação de serviços às necessidades individuais de diferentes clientes (service customization)* : agentes móveis podem-se mover para o local de um certo serviço e implementar (servir de) uma nova interface para aquele serviço, fornecendo uma funcionalidade específica para as necessidades de (um conjunto de) clientes. Diferentes grupos de usuários podem, com isto, ter diferentes interfaces de um mesmo sistema remoto, cada uma atendendo a necessidades específicas do grupo. O projeto da interface daquele serviço pode então ser feito de forma mais simples [White 94b], fornecendo apenas funcionalidades mais básicas;

- *suporte à operação desconectada* : como uma conseqüência do primeiro item acima, agentes móveis suportam a execução de atividades envolvendo dispositivos móveis, que podem operar desconectados da rede. Depois de se

lançar um agente móvel para executar uma aplicação distribuída, o dispositivo móvel pode ser desconectado ou desligado.

Uma alternativa seria submeter a execução de atividades distribuídas que deveriam ser feitas independentemente do sistema original para um outro sistema, que estaria então conectado à rede, por exemplo, uma *estação de suporte* (*supporting station*) em ambientes de computação móvel [Elmagarmid, Jing, Furukawa 95]. Agentes móveis tornam, no entanto, a existência de um tal sistema (ou sistemas) desnecessária. Além disto, submeter um grande número de atividades distribuídas para o mesmo conjunto de sistemas conectados à rede pode comprometer a extensibilidade da aplicação (aqueles sistemas podem passar a representar um gargalo).

Adicionalmente, agentes móveis podem ser carregados *em* dispositivos móveis, para continuar executando atividades no dispositivo móvel *enquanto* o dispositivo está desconectado;

- *flexibilidade no gerenciamento de configuração de software em um sistema distribuído*: agentes móveis representam uma forma simples de se instalar e desinstalar código em um nó remoto. O movimento de um agente para um nó corresponde a se instalar um programa (o agente) naquele nó e o movimento de saída do nó corresponde a desinstalá-lo. Pode-se também fazer *cache* de agentes em agências, tornando o transporte do código do agente desnecessário quando mais tarde (dentro de um certo período de tempo) agentes com o mesmo código movam para aquela agência. O código do agente pode também permanecer instalado em uma agência até ser explicitamente desinstalado. A instalação remota de código é executada automaticamente, sem necessidade de intervenção da administração do nó remoto.

Conceitos alternativos podem ser considerados que forneceriam algumas das vantagens listadas acima. Agentes móveis representam um conceito que pode fornecer todos estes benefícios potenciais *com uma única tecnologia.*

Os benefícios que podem ser explorados com agentes móveis os tornam um paradigma adequado para ser usado no projeto de sistemas distribuídos para ambientes abertos (*open systems*) [Mendes, Assis Silva 99]. As propriedades de ambientes abertos os tornam um exemplo de ambiente onde as necessidades de se usar soluções que forneçam tais benefícios são mais relevantes. Por exemplo, sistemas distribuídos que devem ser altamente extensíveis em um ambiente massivamente distribuído, heterogêneo e multi-organizacional parecem beneficiar-se muito de agentes móveis, principalmente devido a sua habilidade de descentralizar o processamento, adaptar-se à autonomia de sistemas, permitir flexibilidade no gerenciamento de código instalado e de suportar bem interações com usuários humanos com os benefícios de execução local. O uso efetivo de agentes móveis em ambientes abertos, no entanto, requer que soluções para alguns problemas sejam encontradas, principalmente em relação a segurança e tolerância a falhas.

5. Alguns Sistemas de Agentes Móveis

O primeiro trabalho que trata a noção de agentes móveis parece ser [Tsichritzis 85]. Neste artigo o autor descreve o conceito de objetos chamados *Kno*. Um objeto *Kno* é um objeto que tem a capacidade de autonomamente migrar através de um ambiente distribuído e possui inteligência. A primeira descrição de um sistema com as características de um agente móvel aparece em [Wolfson, Voorhees, Flatley 89], onde o conceito de *roteadores inteligentes* (*intelligent routers*) é descrito. Agentes móveis ganharam mais atenção com Telescript em 1994 [White 94a][White 94b]. Depois desta época, muitos projetos em centros de pesquisa e produtos na indústria começaram a surgir. Nesta seção será apresentado um resumo de alguns dos principais sistemas de agentes móveis.

5.1. Roteadores Inteligentes (*Intelligent Routers*)

O sistema de Roteadores Inteligentes [Wolfson, Voorhees, Flatley 89] era desenvolvido pela Siemens. O objetivo deste sistema era suportar migração, solicitada pela aplicação, de uma unidade de execução de um nó para outro em um ambiente distribuído, para fazer uso de recursos disponíveis no nó remoto. *Roteadores inteligentes* foi o nome dado às entidades migrantes. Os requisitos definidos para o sistema eram análogos aos de sistemas de agentes móveis atuais.

O sistema deveria suportar diferentes tipos de linguagens para se programar roteadores inteligentes. Migração fraca ou forte seria suportada dependendo-se do suporte fornecido pela linguagem de programação utilizada. Linguagens interpretadas eram consideradas como um meio de se suportar a execução de roteadores inteligentes em um ambiente heterogêneo. A linguagem de programação de roteadores deveria também definir mecanismos específicos para se suportar a comunicação entre roteadores. No protótipo descrito, roteadores inteligentes eram programados em uma linguagem simples chamada MPL.1 (*Manufacturing Programming Language 1*), orientada à principal aplicação alvo do sistema, sistemas de manufatura. O protótipo não suportava comunicação entre roteadores inteligentes. Suporte a tolerância a falhas já estava planejado utilizando-se uma plataforma de comunicação em grupo, mas nenhum enfoque específico foi descrito.

O estado corrente deste projeto não está descrito na literatura.

5.2. Telescript

Com Telescript [White 94a][White 94b], General Magic apresentou o conceito de objetos ativos migrantes como um conceito adequado para se suportar a distribuição de componentes de aplicação em um ambiente distribuído, onde os computadores podem pertencer a organizações diferentes. O uso de agentes móveis é sugerido para se fornecer melhor desempenho (*performance*) de aplicações distribuídas e (principalmente) adaptação de interfaces em sistemas remotos a necessidades individuais de usuários (*customization*), especialmente sobre redes públicas.

Agentes móveis eram programados na *linguagem Telescript*, uma linguagem proprietária desenvolvida pela General Magic. A linguagem Telescript é orientada a objetos, interpretada e persistente. Telescript suportava migração forte. Um agente se move invocando uma operação *go*, que pode ser invocada em qualquer ponto do

programa do agente. Um agente executa em *lugares*. Lugares podem ser aninhados e o conceito de região como um conjunto de lugares pertencentes a uma mesma organização é suportado. Agentes móveis podem comunicar uns com os outros ou localmente *encontrando-se* (*meeting*) em um lugar ou remotamente, através do estabelecimento de uma conexão.

Telescript definiu uma maneira específica de se codificar agentes para transmissão. O transporte de agentes é controlado por um protocolo específico, chamado *protocolo de interconexão de plataformas* (*platform interconnect protocol* - PIP). Este protocolo controla o transporte do agente, incluindo medidas de segurança. Segurança é fornecida através de *credenciais* e *permissões*. Permissões são usadas para se modelar restrições ao conjunto de instruções que um agente pode usar ou ao conjunto de recursos que ele pode acessar em um lugar e para modelar até que quantidade um certo recurso pode ser usado. Restringindo-se e controlando-se o uso de recursos por um agente, permissões servem a dois propósitos: proteger o ambiente contra agentes maliciosos e proteger usuários contra agentes mal programados que poderiam inadvertidamente mover através da rede usando uma quantidade não desejada de recursos (o usuário pode ter que pagar pelo uso de tais recursos).

Telescript era baseado em hardware especial para suportar persistência de agentes. Telescript não é desenvolvido mais.

5.3. Agent Tcl

Agent Tcl [Gray 95][Gray, Cybenko et al. 96], desenvolvido pelo Dartmouth College, Inglaterra, suporta agentes programados em uma versão estendida da linguagem Tcl. O sistema é, no entanto, projetado para suportar múltiplas linguagens. Agent Tcl suporta migração forte. Migração forte é implementada modificando-se o interpretador Tcl (Tcl core).

O sistema suporta comunicação entre agentes através ou de mensagens ou *conexões diretas*. Uma mensagem é uma seqüência arbitrária de bytes sem sintaxe ou semântica pré-definida, exceto para dois tipos especiais de mensagens: *mensagens de eventos*, para se fornecer comunicação assíncrona de uma ocorrência importante, e *mensagens de conexão*, para se requisitar ou rejeitar o estabelecimento de uma conexão direta. Uma conexão direta é um *stream* de mensagens entre dois agentes e fornece principalmente uma forma mais eficiente de comunicação. Comunicação pode ser feita por agentes em diferentes locais. Durante uma conexão, os agentes comunicantes não se podem mover. Para se enviar mensagens ou construir conexões, o emissor deve especificar o local onde o agente recebedor deve estar.

O sistema continuou a ser desenvolvido para incorporar: uma implementação de eventos; múltiplas linguagens (incluindo linguages funcionais, declarativas e imperativas); espaços de nomes hierárquicos; armazenamento não volátil para o estado de agentes; segurança; formas de se definir uma estratégia de migração utilizando informação sobre a rede (*network awareness*); e mecanismos para se permitir a continuação da comunicação com um agente mesmo enquanto ele estiver se movendo. Além disto, o conceito de *docks* está sendo desenvolvido para suportar operação desconectada. Se a máquina para onde um agente quer se mover não está alcançável, o agente é armazenado em um *dock* na rede. O agente é então enviado para a máquina desejada quando ela se tornar disponível.

Este sistema está sendo desenvolvido agora com o nome de *D'Agents*.

5.4. TACOMA

O projeto TACOMA [Johansen, van Renesse, Schneider 95a][Johansen, van Renesse, Schneider 95b], desenvolvido pela Universidade de Tromsø e Universidade de Cornell, tem como um de seus objetivos suportar várias linguagens de programação de agentes, interpretadas ou compiladas. As versões atuais do TACOMA suportam agentes programados em C, C++, ML, Perl, Python, Scheme e Visual Basic [Johansen, Schneider, van Renesse 98]. TACOMA suporta migração fraca. O programador tem que explicitamente especificar o que representa o estado do agente.

O sistema TACOMA é baseado na noção de *folders, briefcases, file cabinets* e na operação *meet*. Um *folder* é uma unidade de dados acessível a agentes. Um agente pode armazenar e ler itens de dados de/para um *folder*. Um *briefcase* é uma coleção de *folders* associado a um agente. Quando um agente se move, ele carrega uma *briefcase* com ele. *File cabinets* são coleções de *folders* tais como *briefcases*, mas *file cabinets* não se movem. Eles representam repositórios de dados permanentes.

TACOMA possui uma operação *meet*, que representa a abstração básica de interação entre agentes. Dois agents devem-se "encontrar" ("*meet*") localmente em algum local (chamado *lugar* em TACOMA) para comunicar. Durante a operação *meet*, um agente transfere a informação desejada para o outro agente dentro de uma *briefcase*.

Para um agente migrar para um outro lugar, o código e dados do agente são colocados em uma *briefcase* e uma operação *meet* é executada, especificando-se um agente no sistema remoto que pode executar o código daquele agente. *Folders* especiais existem para armazenar tipos especiais de dados. Por exemplo, o *folder* CODE é usado para armazena o código de um agente e o *folder* HOST especifica onde a operação *meet* deve ser executada.

A operação *meet* suporta mobilidade e comunicação. Uma forma especial desta operação existe para se executar RPC.

TACOMA fornece também tolerância a falha de agentes, com a noção de *rear guards* [Johansen, Marzullo et al. 98]. *Rear guards* representam agentes que são deixados para trás enquanto o agente se move de uma agência para outra. *Rear guards* podem recuperar a execução de agentes no caso de falhas de agências de longa duração.

5.5. Aglets

O sistema Aglets [Lange 97][Lange, Chang 96][Oshima, Karjoth, Ono 98] é desenvolvido pela IBM do Japão. O sistema suporta migração fraca e Java é a linguagem de programação de agentes. Aglets é completamente programado em Java.

Um agente móvel é chamado de um *aglet* no sistema. Um aglet executa em um *contexto*. Um contexto fornece um ambiente uniforme para os aglets e fornece a eles uma interface para interagirem com o ambiente (por exemplo, para criar outros aglets). O contexto protege o ambiente contra aglets maliciosos. Um aglet pode requisitar seu movimento para um outro nó. Um nó pode também solicitar o carregamento *(fetch*) de um aglet de um nó remoto.

A comunicação entre aglets é feita através de troca de mensagens. Um aglet comunica com outro aglet trocando objetos da classe *Message*. A troca de mensagens pode ser feita síncrona ou assincronamente (com resultados futuros ou não). Prioridades podem ser atribuídas a mensagens. A comunicação com um aglet é feita através de seu *aglet proxy*. Um proxy serve como um representante do agente. Os proxies executam um controle de segurança que protege o aglet contra uso malicioso de seus métodos e fornecem uma interface uniforme para comunicação com o aglet, independentemente de o aglet recebedor estar no mesmo nó que o aglet que envia a mensagem ou não. Aglet proxies *não* mantêm informação atualizada da localização de agentes enquanto eles migram. Se um aglet se move, um proxy que o referenciava anteriormente se torna inválido (versão 1.1).

O sistema Aglets fornece uma forma de segurança baseada em domínios. Todos os servidores aglet (agências) no mesmo domínio consideram-se confiáveis mutuamente e eles compartilham uma chave secreta. Quando um aglet é enviado de um servidor aglet para um outro, cada servidor comunicante é capaz de verificar se o outro servidor pertence ao mesmo domínio e então decide se ele é confiável ou não. Um servidor aglet confia nas credenciais de um aglet se elas foram emitidas por um servidor do mesmo domínio. Durante a comunicação, um código de integridade é computado para cada mensagem usando a chave secreta do domínio e este código é enviado com a mensagem. Já que o código de integridade somente pode ser gerado por um servidor do mesmo domínio, o recebedor da mensagem pode determinar se a mensagem foi enviada por um servidor do mesmo domínio ou não e pode então aceitar a mensagem ou rejeitá-la.

A autorização de acessos feitos por um aglet a recursos de um servidor aglet é baseada no uso do mecanismo de segurança de Java. Permissões são atribuídas ou não a um aglet de acordo com o seu *codebase* e seu dono.

5.6. Mole

O sistema Mole [Baumann, Hohl et al. 97] é desenvolvido pela Universidade de Stuttgart. Mole suporta migração fraca e Java é a linguagem de programação de agentes. O ambiente onde um agente executa é chamado em Mole de um *lugar.*

Mole suporta comunicação através de troca de mensagens, RPC e comunicação anônima. Comunicação através de mensagens e RPC são orientadas a sessão.

Uma sessão é um relacionamento entre *dois* agentes comunicantes. Para se estabelecer uma sessão, os agentes pares comunicantes têm que ser identificados. Mole suporta identificação através de identificadores de agentes (*agent ids*) e através de *badges*. Um *badge* é uma identificação definida pela aplicação, que um agente pode utilizar. Um *badge* com um identificador "tarefa A", por exemplo, pode ser usado por um agente para indicar que ele está executando parte da tarefa A. Um agente pode ter vários *badges*. Agentes podem então ser identificados por um par (*id_de_lugar*, *predicado*). O *id_de_lugar* identifica um lugar no ambiente. O *predicado* é uma expressão envolvendo *badges* e conectores lógicos. Um agente é identificado por um tal par se ele está no lugar com identificador *id_de_lugar* e se os *badges* que ele atualmente está usando satisfazem *predicado*.

Sessões podem ser intra- ou inter-lugares. Durante uma sessão, um agente não pode se mover. Se ele se move, a sessão termina implicitamente. Duas operações são

oferecidas para se estabelecer conexões: *PassiveSetUp* e *ActiveSetUp*. *PassiveSetUp* é usada por um agente para indicar que ele quer participar de uma sessão. *ActiveSetUp* é usada para se emitir um pedido de estabelecimento de conexão síncrono, ou seja, o agente fica bloqueado até que ou a sessão é estabelecida com sucesso ou um *timeout* ocorra. Se um predicado, como descrito no parágrafo anterior, é usado para identificar possíveis parceiros de comunicação e mais de um agente satisfizer o predicado, um único agente é escolhido aleatoriamente para completar a conexão.

Comunicação durante uma sessão é feita através de objetos *com*. Dois tipos de objetos *com* existem: um *objeto RMI* e um *objeto Messaging*. Com um objeto RMI, métodos de um agente remoto podem ser invocados. Através de um objeto *Messaging*, mensagens podem ser enviadas assincronamente entre os participantes da sessão.

Comunicação de eventos é feita utilizando-se *objetos de sincronização*. Um objeto de sincronização implementa uma regra do tipo *evento, condição, ação*. Um objeto de sincronização recebe eventos de agentes produtores. A parte de condição de uma regra é uma expressão lógica envolvendo tipos de eventos e informação de estado do objeto de sincronização. Se a condição se torna verdadeira, a parte de ação é executada. Regras podem também ser especificadas para que a parte de ação execute depois que um período específico de tempo for transcorrido.

Segurança de lugares é feita usando o mecanismo de segurança embutido de Java. Os lugares podem decidir quais agentes eles irão aceitar, baseado no lugar onde o agente foi criado. Adicionalmente, um mecanismo de segurança está sendo desenvolvido para proteger agentes contra servidores Mole maliciosos. O mecanismo combina técnicas de ofuscação de código com um período de timeout (veja seção 3.6).

O sistema Mole suporta contabilização dos recursos citados na seção 3.7. No contexto do projeto, foram desenvolvidos um algoritmo para detecção de órfãos [Baumann, Rothermel 98] e um algoritmo para se garantir a propriedade *exatamente-uma-vez* (*exactly-once*) para a execução de agentes móveis [Rothermel, Straßer 98].

5.7. Voyager

Voyager [ObjectSpace 97][ObjectSpace 98b] é um produto em desenvolvimento pela ObjectSpace, Inc. Voyager suporta migração fraca e Java é a linguagem de programação de agentes.

Comunicação em Voyager é suportada através de objetos proxy. Para que um objeto possa interagir com um objeto remoto, um objeto proxy relativo ao objeto remoto é construído localmente. O objeto remoto e o objeto proxy implementam a mesma interface. A comunicação é feita invocando-se métodos do objeto proxy. Se o objeto remoto emite uma exceção, a exceção é encaminhada ao chamador, como se a exceção tivesse sido emitida localmente. Classes de proxies são geradas automaticamente. Com proxies, objetos podem também ser criados remotamente.

Objetos proxy mascaram o movimento de agentes. Métodos de um agente podem ser chamados mesmo enquanto ele se move (transparência de localização/movimento). Esta transparência é obtida pelo sistema deixando-se *secretárias* em cada sistema por onde o agente passa. Uma secretária que recebe uma mensagem para um agente encaminha a mensagem para o próximo sistema para onde o agente se moveu. No próximo sistema uma outra secretária existirá, se o agente já se tiver movido. As

secretárias ao longo do caminho que o agente percorreu encaminham assim a mensagem até que ela alcance o agente. As secretárias são apagadas pelo processo de *coleta de lixo* (*garbage collection*). Tão logo nenhuma referência para uma secretária exista mais, a secretária será coletada e assim toda a seqüência. Se o agente remoto retornar um resultado, a mensagem de resultado é carimbada com a nova localização do agente, de forma que o objeto proxy pode atualizar seu conhecimento sobre a localização do objeto remoto.

Além de mensagens síncronas, Voyager suporta também mensagens assíncronas sem esperar resultado (mensagens *oneway*), mensagens assíncronas com valores de retorno (mensagens *future*) e mensagens dinâmicas. Voyager suporta o conceito de *espaço* (*space*) descrito na seção 3.

Cada agente tem um tempo de vida determinado. Um objeto pode viver para sempre ou definir seu tempo de vida com base no tempo em que o objeto já existe ou está inativo ou em um determinado instante no tempo. Quando um agente atinge seu tempo de vida, ele morre e é automaticamente eliminado pelo mecanismo de coleta de lixo.

Persistência de agentes é suportada em Voyager por uma classe que fornece métodos para se obter e armazenar os estados de objetos (classe *Snapshot*) e por um esquema de *ativadores* (*Activator framework*). Cada programa Voyager tem um *gerenciador de ativadores* junto ao qual um conjunto de *ativadores* são registrados. Cada ativador pode representar uma interface para um banco de dados que pode tratar o armazenamento e recuperação de objetos. Quando uma mensagem chega para um objeto que está no banco de dados, o gerenciador de ativadores interage com o ativador do objeto para reinstanciar o objeto na memória principal. A decisão de como e quando objetos são armazenados no banco de dados é deixada para o programador dos agentes.

Segurança em Voyager é suportada com o mecanismo de segurança embutido de Java. Voyager fornece um próprio gerenciador de segurança. Em Voyager, classes carregadas através do CLASSPATH local são consideradas “nativas“. Outras classes são consideradas “estranhas“. Classes nativas podem executar qualquer operação. Classes estranhas possuem acesso restrito, considerando-se cada operação caso a caso.

Voyager fornece também um serviço de transações [ObjectSpace 98a].

5.8. Grasshopper

Grasshopper [IKV++ 98] é hoje desenvolvido pela IKV++ (inicialmente foi desenvolvido pelo instituto GMD FOKUS). Grasshopper fornece migração fraca e Java é a linguagem de programação de agentes.

Grasshopper suporta a noção de regiões, agências e lugares, como definido na seção 3. Em cada região há um *banco de registros da região* (*region registry*). O banco de registros fornece informação para se localizar agências, lugares e agentes/serviços existentes na região. Para que estas entidades possam ser encontradas, elas devem ser registradas no banco de registros da região. Para entidades não móveis, o procedimento de registro é feito uma vez apenas. Para agentes móveis, o procedimento de registro é feito a cada vez que o agente se move, para que a informação sobre a localização atual do agente seja mantida atualizada nos bancos de registros.

Em Grasshopper, comunicação entre agentes é feita através de objetos proxy. Proxies fornecem transparência de localização em Grasshopper, usando-se os bancos de

registros. Na versão 1.1, descrita nesta seção, o mecanismo de suporte a interações entre bancos de registros de regiões não está implementado ainda. Por isto, comunicação com transparência de localização é restrita a apenas uma região. Comunicação pode ser feita através de IIOP (*Internet Inter-ORB Protocol*), Java RMI ou utilizando-se conexões através de *socket*. Uma forma de multicast não confiável (*non reliable*) é também fornecida [Leichsenring 98]. O multicast é implementado através de um *proxy de grupo*. Um proxy de grupo mantém informação sobre um grupo de objetos. Através de um proxy de grupo uma mensagem é distribuída para os membros do grupo e os resultados da mensagem (se houver) são coletados. Um cliente que envia uma mensagem em multicast pode ficar bloqueado esperando por resultados ou não. Ele pode esperar até que ou todos os resultados cheguem ou até que o primeiro resultado chegue. Se o cliente não bloqueia, ele pode periodicamente perguntar ao proxy de grupo se os resultados do multicast já estão disponíveis ou não.

Segurança em Grasshopper é classificada em *externa* e *interna*. Segurança externa se refere a aspectos de segurança relacionados à comunicação entre agências e bancos de registros de regiões. Segurança interna se refere à proteção de agências contra acessos não autorizados de agentes e para proteger um agente de outros agentes. Segurança externa é fornecida para interações através de Java RMI ou conexões através de *socket*, utilizando-se SSL (*Secure Socket Layer*). Segurança interna é baseada no mecanismo de segurança de Java. Código sensitivo a segurança pode ser indicado utilizando-se um método especial. Quando este método é executado, uma checagem é feita para se verificar se o chamador pode continuar executando o código depois deste método ou não.

O serviço de persistência de Grasshopper fornece métodos para um agente requisitar que seu estado seja armazenado e para se recuperar o estado do disco. Se um agente estiver presente no banco de dados e for contactado por outro agente, o agente pode ser automaticamente carregado na memória. O serviço de persistência é baseado na serialização de objetos.

Grasshopper implementa a proposta MASIF [OMG 98]. O sistema suporta ambas as interfaces definidas pela proposta: *MAFAgentSystem* e *MAFFinder*. A primeira define métodos para suporte a tarefas de gerenciamento de agentes, tais como suspender a execução de um agente, retomar a sua execução, criar agentes, entre outras. Através desta interface pode-se interagir com agências. A outra interface fornece um serviço de nomes para se registrar e encontrar agentes, agências e lugares, ou seja, para se interagir com bancos de registros.

Bibliografia

[Alonso, Kamath et al. 94] G.Alonso, M.Kamath, D.Agrawal, A.El Abbadi, R.Günthör, C.Mohan. Failure Handling in Large Scale Workflow Management Systems. *IBM Research Report RJ9913*. November, 1994

[Assis Silva 99] F.M.Assis Silva. *A Transaction Model based on Mobile Agents*. Tese de Doutorado. Universidade Técnica de Berlim, Alemanha. 1999

[Assis Silva, Popescu-Zeletin 98] F.M. Assis Silva, R.Popescu-Zeletin. An Approach for Providing Mobile Agent Fault Tolerance. Proceedings of the Second International Workshop, MA'98. Stuttgart, Germany. September, 1998. *Lecture Notes in Computer Science 1477*. K.Rothermel, F.Hohl (eds). Springer-Verlag. 1998. pp.14-25

[Baumann, Hohl et al. 97] J.Baumann, F.Hohl, K.Rothermel, M.Straßer. Mole - Concepts of a Mobile Agent System. *Bericht 1997/15.* Universität Stuttgart - Fakultät Informatik. August 1997

[Baumann, Rothermel 98] J.Baumann, K.Rothermel. The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. Proceedings of the Second International Workshop, MA'98. Stuttgart, Germany. September, 1998. *Lecture Notes in Computer Science 1477*. Springer-Verlag. 1998. pp. 2-13

[Busse, Covaci, Nietfeld 98] I.Busse, S.Covaci, M.Nietfeld. Sicherheitsrisiken bei Mobilen Agenten und deren Lösung in einer Java basierten Plattform. *Praxis der Informationsverarbeitung und Kommunikation - 3/98.* K.G.Saur-Verlag. Juli-September, 1998

[Cabri, Leonardi, Zambonelli 98] G.Cabri, L.Leonardi, F.Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. Proceedings of the Second International Workshop, MA'98. Stuttgart, Germany. September, 1998. *Lecture Notes in Computer Science 1477*. Springer-Verlag. 1998. pp. 237-248

[Cai, Gloor, Nog 96] T.Cai, P.A.Gloor, S.Nog. DartFlow: A Workflow Management System on the Web using Transportable Agents. *Technical Report PCS-TR96-283.* Department of Computer Science. Dartmouth College. 1996

[Cardelli 95] L.Cardelli. A Language with Distributed Scope. *Computing Systems*. Vol.8, No.1. 1995. pp.27-59

[Chess, Harrison, Kershenbaum 94] D.Chess, C.Harrison, A.Kershenbaum. Mobile Agents: Are They a Good Idea?. *IBM Technical Report RC 19887.* October, 1994

[Elmagarmid, Jing, Furukawa 95] A.Elmagarmid, J.Jing, T.Furukawa. Wireless Client/Server Computing for Personal Information Services and Applications. *ACM SIGMOD Record*. December, 1995.

[Finin, Labrou, Mayfield 95] T.Finin, Y.Labrou, J.Mayfield. KQML as an Agent Communication Language. *Software Agents*. J.Bradshaw (ed.). MIT Press. Cambridge. 1995

[Franklin, Graesser 96] S.Franklin, A.Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag. 1996

[Freier, Karlton, Kocher 96] A.O.Freier, P.Karlton, P.C.Kocher. *The SSL Protocol - Version 3.0*. Internet Draft, IETF. November, 1996. *ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf*

[Fuggetta, Picco, Vigna 98] A.Fuggetta, G.P.Picco, G.Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering.* Vol.24, No.5. May, 1998. pp.342-361

[Fünfrocken 98] S.Fünfrocken. Transparent Migration of Java-based Mobile Agents: Capturing and Reestablishing the State of Java Programs. Proceedings of the Second International Workshop, MA'98. Stuttgart, Germany. September, 1998. *Lecture Notes in Computer Science 1477*. Springer-Verlag. 1998. pp.26-37

[Gong 98] L.Gong. *Java Security Architecture (JDK 1.2)*. Sun Microsystems Inc. March, 1998. Disponível em: *http://www.javasoft.com/products/jdk/1.2/docs/guide/security/*

[Gosling, McGilton 96] J.Gosling, H.McGilton. *The Java Language Environment - A White Paper.* SUN Microsystems. May, 1996. Disponível em: *http://java.sun.com/doc/language_environment/*

[Gray 95] R.S.Gray. Agent Tcl: A transportable agent system. *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM95)*. Baltimore, Maryland. December, 1995

[Gray, Cybenko et al. 96] R.Gray, G.Cybenko, D.Kotz, D.Rus. *Agent Tcl.* May, 1996. Disponível em: *http:// www.cs.dartmouth.edu/papers/index.html*

[Howell 98] J.Howell. Straightforward Java Persistence through Checkpointing. *Proceedings of the Third International Workshop on Persistence and Java(tm) (PJW3)*. USA. September, 1998. Disponível em: *http://www.sunlabs.com/research/forest/com.sun.labs.pjw3.main.html*

[IKV++ 98] IKV++. *Grasshopper - An Intelligent Mobile Agent Platform - Volume 2 - Programmer's Guide - Version 1.1*. IKV++. 1998

[ISO/IEC 94] ISO/IEC. *Information Technology - Open Distributed Processing - Reference Model - Open Distributed Processing - Part 1 - Overview and Guide to Use - ISO/IEC JTC 1/SC 21.* September 7th. 1994

[ITU-T 95] ITU-T. *Open Distributed Processing - Reference Model - Part 3: Architecture.* ITU-T. February, 1995

[Johansen, Marzullo et al. 98] D.Johansen, K.Marzullo, F.B.Schneider, K.Jacobsen, D.Zagorodnov. NAP: Practical Fault-Tolerance for Itinerant Computations. *Technical Report TR98-1716*. Department of Computer Science, Cornell University. USA. November, 1998

[Johansen, Schneider, van Renesse 98] D.Johansen, F.B.Schneider, R.van Renesse. What TACOMA Taught Us. *Mobility, Mobile Agents and Process Migration - An Edited Collection*. D.Milojicic, F.Douglis, R.Wheeler (eds). Addison-Wesley Publishing Company, 1998

[Johansen, van Renesse, Schneider 95a] D.Johansen, R.van Renesse, F.B.Schneider. An Introduction to the TACOMA Distributed System - Version 1.0. *Computer Science Technical Report 95-23*. Institute of Mathematical and Physical Sciences - Department of Computer Science - University of Tromsø. June 1995

[Johansen, van Renesse, Schneider 95b] D.Johansen, R.van Renesse, F.B.Schneider. Operating System Support for Mobile Agents. *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*. 1995

[Jordan, Atkinson 98] M.Jordan, M.Atkinson. Orthogonal Persistence for Java - A mid-term Report. *Proceedings of the Third International Workshop on Persistence and Java(tm) (PJW3)*. USA. September, 1998. Disponível em: *http://www.sunlabs.com/research/forest/com.sun.labs.pjw3.main.html*

[Knabe 95] F.C.Knabe. *Language Support for Mobile Agents*. PhD Thesis. Carnegie Mellon University. Pittsburgh, PA, USA. December, 1995

[Lange 97] D.B.Lange. *Java Aglet Application Programming Interface (J-AAPI) White Paper - Draft 2.* IBM Tokyo Research Laboratory. February, 1997. Disponível em: *http://www.trl.ibm.co.jp/ aglets/JAAPI-whitepaper.html*

[Lange, Chang 96] D.B.Lange, D.T.Chang. *IBM Aglets Workbench - Programming Mobile Agents in Java - A White Paper - Draft.* IBM Corporation. September, 1996. Disponível em: *http:// www.trl.ibm.co.jp/aglets/whitepaper.htm*

[Leichsenring 98] A.Leichsenring. *Development of a Basic Agent Communication Service for the Grasshopper Platform*. Diplomarbeit. Technical University Berlin. 1998

[Lingnau, Drobnik 96] A.Lingnau, O.Drobnik. Making Mobile Agents Communicate: A Flexible Approach. *Proceedings of the 1st Annual Coneference on Emerging Technologies and Applications in Communications (etaCOM'96)*. USA. May, 1996

[Magedanz, Eckardt 96] T.Magedanz, T.Eckardt. Mobile Software Agents: A new Paradigm for Telecommunications Management. *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Kyoto, Japan. April, 1996

[Magedanz, Popescu-Zeletin 96] T.Magedanz, R.Popescu-Zeletin. Towards "Intelligence on Demand" - On the Impacts of Intelligent Agents on IN. *Proceedings of the 4th International Conference on Intelligence in Networks*. Bordeaux, France. November, 1996

[Mendes, Assis Silva 99] M.Mendes, F.M.Assis Silva. Mobile Agents in Autonomous Decentralized Systems. Artigo para o painel de discussão sobre Autonomous Agents. *Proceedings of the Fourth International Symposium on Autonomous Decentralized Systems (ISADS'99)*. Tokyo, Japao. 21-23 de Março, 1999

[Mendes, Loyolla et al. 96] M. Mendes, W. Loyolla, T. Magedanz, F.M. Assis Silva, S. Krause. Agent Skills and their Roles in Mobile Computing and Personal Applications. *Proceedings of the IFIP 14th World Computer Congress, World Conference on Mobile Communications*. Australia. September, 1996

[Merz, Liberman et al. 96] M.Merz, B.Liberman, K.Müller-Jones, W.Lamersdorf. Interorganizational Workflow Management with Mobile Agents in COSM. *Proceedings of the* PAAM '96. London. April, 1996

[Nwana 96] H.S.Nwana. Software Agents: An Overview. *Knowledge Engineering Review*. Vol.11, No.3. October/November, 1996. pp.205-244

[ObjectSpace 97] ObjectSpace. *Voyager Core Technology User Guide - Version 1.0.0.* ObjectSpace, Inc. 1997. Disponível em: *http://www.objectspace.com/developers/voyager/white/index.html*

[ObjectSpace 98a] ObjectSpace. *Voyager Transaction Service Technical Overview*. ObjectSpace, Inc. January, 1998. Disponível em: *http://www.objectspace.com/developers/voyager/white/index.html*

[ObjectSpace 98b] ObjectSpace. *Voyager Core Technology 2.0 - User Guide*. ObjectSpace, Inc. 1998. Disponível em: *http://www.objectspace.com/developers/voyager/white/index.html*

[OMG 92] OMG. *Object Management Architecture Guide - Revision 2.0.* Second Edition. September, 1. 1992

[OMG 98] OMG. *Mobile Agent System Interoperability Facility (MASIF) - TC Document orbos/98-03-09*. OMG. March, 1998. Disponível em: *ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf*

[Oshima, Karjoth, Ono 98] M.Oshima, G.Karjoth, K.Ono. *Aglets Specification 1.1 Draft.* IBM Corporation. September, 1998. Disponível em: *http://www.trl.ibm.co.jp/aglets/spec11.html*

[Ousterhout 95] J.Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley. 1995

[Rothermel, Straßer 98] K.Rothermel, M.Straßer. A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents. *Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS'98)*. West Lafayette, USA. October, 1998. pp. 100-108

[Schneider 97] F.B.Schneider. Towards Fault-tolerant and Secure Agentry. *Proceedings of the 11th International Workshop on Distributed Algorithms*. Saarbrücken, Germany. September, 1997

[Tsichritzis 85] D.Tsichritzis. Objectworld. *Office Automation*. Springer-Verlag. 1985

[Veijalainen, Eliassen, Holtkamp 92] J.Veijalainen, F.Eliassen, B.Holtkamp. The S-transaction Model. *Database Transaction Models for Advanced Applications*. A.K.Elmagarmid (ed.). Capítulo 12. Morgan Kaufmann Publishers. USA. 1992. pp.467-513

[Vogler, Kunkelmann, Moschgath 97] H.Vogler, T.Kunkelmann, M.-L.Moschgath. An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions. *Proceedings of the 1997 International Conference on Parallel and Distributed Systems (ICPADS'97)*. 1997. pp.268-274

[White 94a] J.E.White. *Telescript Technology: Scenes from the Electronic Marketplace*. General Magic. 1994

[White 94b] J.E.White. *Telescript Technology: The Foundation for the Electronic Marketplace*. General Magic. 1994

[Wolfson, Voorhees, Flatley 89] C.D.Wolfson, E.M.Voorhees, M.M.Flatley. Intelligent Routers. *Proceedings of the 9th International Conference on Distributed Computing Systems*. California. June, 1989. pp. 371-376