DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9805

Agentes Móveis: um Tutorial

Markus Endler

dezembro de 1998

Agentes Móveis: um Tutorial

Markus Endler
Departamento de Ciência da Computação
IME-Universidade de São Paulo
São Paulo, Brazil
E-mail: endler@ime.usp.br

Resumo

Este artigo é um tutorial sobre sistemas de agentes móveis, que descreve os principais conceitos, componentes da infra-estrutura necessária, exemplos de ambientes de programação existentes, e as possíveis formas de interação entre agentes. Inicialmente discutiremos o que são agentes móveis, como surgiram e quais são as possíveis aplicações. Em seguida discutiremos os principais conceitos e as propriedades comuns de ambientes de desenvolvimento de agentes móveis. Depois apresentaremos as principais componentes, características e exemplos de programação de alguns sistemas específicos. Concluiremos o tutorial com a nossa visão do futuro desta tecnologia.

1 Introdução

Nos últimos anos têm havido indícios de que está ocorrendo uma mudança no paradigma de programação distribuída. Esta mudança vem sendo alimentada pela crescente proliferação do uso da rede mundial internet para as mais diversas aplicações e o aparecimento de novas tecnologias que, entre outras coisas, estão viabilizando a execução de programas em sistemas distribuídos heterogêneos. Como exemplo concreto de uma destas tecnologias inovadoras, podemos citar o modelo de execução da linguagem Java, que tem como uma de suas mais importantes características a portabilidade de seu código gerado (byte-code) e a capacidade de carregamento dinâmico de classes.

A principal mudança de paradigma está relacionada à capacidade de transferência de código executável entre computadores interligados por uma rede. Até agora o principal paradigma da programação distribuída era o da transferência de dados representando os argumentos e os resultados da execução de uma função remota. Com este novo paradigma, também denominado de Programação Remota (*Remote Programming*), em vez de um processo cliente enviar os dados para o processamento de uma função implementada somente no servidor, envia-se não só os dados como também o próprio código. Segundo White[12] esta capacidade apresenta uma série de vantagens.

A primeira vantagem é a de que a transferência de dados e código ocorre somente uma vez e em lote, evitando o envio repetido e muitas vezes ineficiente de pequenas quantidades de dados, como ocorre quando há mais de uma chamada para funções remotas por um cliente.

Em segundo lugar, é possível realizar a interação entre a máquina do usuário e a máquina servidora mesmo que o canal de comunicação entre elas tenha conexão temporária, seja instável ou de baixa largura de banda, como ocorre por exemplo no uso da rede telefônica e em particular na telefonia celular. Isto porque, devido à transferência em lote deixa de existir a necessidade de se manter uma conexão virtual entre máquinas durante o processamento.

Uma terceira vantagem diz respeito à adaptabilidade. Em vez do programa cliente ter que ser ligado estaticamente a um código implementando um protocolo específico de acesso às funções remotas

do servidor, como ocorre por exemplo na geração de stubs para a Chamada Remota de Procedimento (RPC), o código enviado ao servidor pode estender as operações básicas do servidor de acordo com as necessidades particulares de cada usuário. Isto também torna mais fácil a atualização destas operações básicas por parte do provedor de serviços.

Com o paradigma de Programação Remota, surgem naturalmente novas formas de interação entre programas que, quando combinadas, criam um leque bastante amplo de padrões de interação inter- e intra-aplicações distribuídas com interessantes características de desempenho, facilidade de sincronização e segurança. O estudo destes padrões, suas características e problemas associados serão também discutidos neste curso.

1.1 O que são agentes móveis?

Agentes móveis essencialmente são programas que têm a capacidade de migrar entre as máquinas de uma rede onde temporariamente executam alguma função, como por exemplo coletar dados locais ou interagir com outros agentes situados na mesma máquina, a fim de resolver uma tarefa global. Em geral, cada agente é dotado de um estado local que tipicamente sofre modificações devido à interação com outros agentes e com o site local, e que geralmente contém dados que determinam o itinerário do agente em sua movimentação na rede.

Agentes móveis já tem sido objeto de pesquisa desde meados dos anos 80, mas recentemente ganharam um grande impulso com o aparecimento da linguagem Java, devido as suas características de independência de plataforma, capacidade de carregamento dinâmico de código e orientação a objeto. No entanto, existem outros ambientes para agentes móveis baseados em outras linguagens interpretadas, como por exemplo a linguagem de scripts Tcl[8].

1.2 Para que servem agentes móveis?

Agentes móveis surgiram da combinação de duas necessidades ortogonais. A primeira veio da área de computação móvel, onde a conexão entre os nós de uma rede é temporária e por vezes instável e onde o paradigma do código móvel é mais adequado do que o paradigma da comunicação por datagrama ou orientada a conexão. A outra se originou da necessidade de assistência na coleta direcionada de dados em uma rede de computadores. Esta tarefa é por vezes chamada de data-mining, e encontra cada vez mais variantes na medida em que redes vão sendo ligadas e informação de diferente natureza está sendo disponibilizada na rede internet.

Além destas, outras motivações também contribuíram para a pesquisa em agentes móveis e aumentaram o leque de possíveis aplicações deste paradigma. Atualmente já existem uma série de aplicações baseadas em agentes móveis. Algumas das mais freqüentes são:

Gerência de Redes A gerência des redes de computadores ainda sofre do problema de que a maioria das decisões de configuração necessita de uma análise detalhada de uma grande quantidade de dados primitivos coletados de dispositivos de rede (roteadores, switches, bridges, etc.). O uso de agentes móveis para a gerência de redes pode facilitar a tarefa do usuário (o gerente de rede) no sentido de permitir uma coleta seletiva e direcionada de dados das componentes de rede, efetuar uma filtragem adequada da informação e eventualmente executar algumas funções de inferência relacionando diferentes eventos a suas possíveis causas. Exemplos de projetos que investigam o uso de agentes móveis para gerência de redes são o *Virtual IP-Network* da FTP Software e *Procura*[10], da Carleton University.

Controle de Segurança Uma outra aplicação relacionada à gerência de uma rede é o controle de segurança. Também neste caso o gerente da rede fica sobrecarregado com tarefas rotineiras de

controle, como a verificação das permissões e user-ids dos processos executando em cada nó na rede, e que visam detectar a presença de usuários intrusos ou da existência de falhas temporárias nos mecanismos de segurança. Neste caso, agentes móveis também podem ser empregados para automatizar a verificação de segurança, através da movimentação por elementos chave da rede, como servidores e roteadores, possivelmante identificando correlações entre anomalias detectadas em diferentes nós.

Ambientes de Apoio à Cooperação Nesta área, imagina-se que agentes possam ser o meio básico de transmissão, controle de acesso e replicação de documentos quaisquer em um ambiente de trabalho cooperativo. Por exemplo, poderia-se imaginar que agentes fossem incubidos de transportar um documento para uma série de nós de uma rede a fim de coletar a assinatura (digital) de diversos usuários, permitir a alteração de um documento conjunto, ou agendar uma reunião entre um grupo de usuários desta rede.

Mercado eletrônico Atrás deste termo genérico mas muito em moda atualmente, aparecem uma série de aplicações reais e potenciais na área de transações comerciais, pesquisa de mercado, análise financeira entre outras. Também aqui a utilidade principal de agentes seria a de percorrer a rede de computadores com a função de procurar provedores de produtos ou serviços, comparar preços e funcionalidade, e eventualmente até com tarefas de estabelecer contatos com outros grupos de usuários. Algumas destas tarefas atualmente já são realizadas pelos ditos *robôs da internet* (também chamados de Softbots, Wanderers ou Spiders)[2] mas que, ao invés de tranferir código executável entre nós, procuram a informação através do carregamento de páginas Web e do percorrimento sistemático de hyperlinks. As aplicações comerciais e financeiras baseadas em agentes móveis ainda constituem uma área emergente, mas é bem provável que em um futuro próximo elas possam vir a se tornar a principal força motriz desta nova tecnologia. Os principais motivos que estão retardando este processo são as complexas questões de segurança e a ausência de uma protocolo comum para a interação de agentes.

2 Conceitos básicos

Nesta seção apresentaremos os principais conceitos encontrados em modelos de agentes móveis. Apesar dos diversos sistemas e ambientes serem baseados em modelos ligeiramente diferentes, tentaremos descrever os conceitos comuns à maioria dos ambientes, identificando similaridades e apontando para eventuais propriedades específicas.

Agente Na sua forma mais primitiva, um agente consiste de código executável e dados a serem executados. Dependendo do modelo de programação, estes podem estar encapsulados em um objeto. Em geral os dados de um agente refletem o seu estado de execução. Em particular, imediatamente antes e depois de uma movimentação de um agente, o seu estado de execução na máquina origem é transformado para uma sequência de bytes, e vice-versa, processo que é chamado de *serialização*. Em alguns ambientes não é possível serializar diretamente o estado completo de um agente uma vez que este involve também um contexto (dados) na camada de suporte à execução. Neste caso, faz-se necessário um armazenamento prévio deste contexto em objetos serializáveis.

Distingue-se ainda entre agentes *estacionários* e *dinâmicos*, sendo que estes últimos podem mudar a sua localização (*lugar*) ao longo da execução. Portanto, além dos dados (o seu estado), um agente tem atributos adicionais que discriminam a sua identidade, Id, e o seu lugar atual, L. A figura 1 mostra o diagrama de possíveis meta-estados de agentes (as elipses) e o efeito das operações mais

comuns sobre agentes (create, clone, move, retract, activate, dispose[1]) Nesta figura, o asterisco (*) denota o conjunto de possíveis estados que um agente pode ter durante o seu período de atividade.

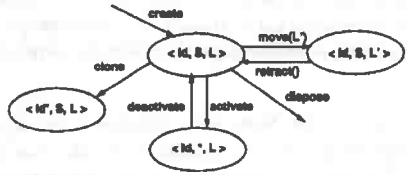

Figura 1: Estados de um agente

Lugar Um *lugar*, *contexto* ou *hospedeiro*, representa uma possível localização de agentes. A cada instante, um agente só pode estar presente em um único lugar. No entanto, um mesmo nó da rede pode implementar vários lugares simultaneamente, sendo a localização física do lugar transparente e irrelevante para o agente. Um lugar tem a possibilidade de recusar a vinda de certos agentes e/ou restringir o acesso destes a certos recursos locais. Em alguns ambientes, como o Obliq[1], as ações dos agentes em cada lugar são determinados por instruções (*briefing*) depositadas em cada lugar para os diferentes agentes que visitarem o lugar.

Servidor O servidor é o programa que precisa estar executando nos nós da rede para que seja possível implementar a movimentação de agentes. Suas principais tarefas são as de recriar um agente a partir de seu estado serializado, ativá-lo, implementar os mecanismos de segurança e autenticação pertinentes e disponibilizar os recursos computacionais (CPU, memória, disco, etc.) para os agentes na forma de um ou mais lugares.

Deslocamento Como mencionado, um agente pode transitar entre diferentes lugares ao longo de sua execução. Isto pode acontecer por iniciativa própria ou por iniciativa de um outro agente. Independente do caso, solicitações de deslocamento são geralmente encaminhadas ao servidor, que então invoca procedimentos preparatórios para a migração. Na maioria das vezes estes procedimentos são métodos do próprio agente a ser migrado, que assim pode determinar o momento adequado para a serialização de seu estado ou inclusive rejeitar um pedido de transferência. Na seção 4.5.3 veremos em mais detalhe este princípio denominado de *callback*, e que é a base de funcionamento de alguns ambientes de programação de agentes móveis.

Encontro Sempre que dois ou mais agentes estão posicionados em um mesmo lugar, e de acordo com suas autorizações, estes podem interagir para fins de troca de dados. Esta interação pode ser direta, através da chamada de métodos do agente parceiro, ou indireta, através do compartilhamento de estado do lugar em questão. Em ambos os casos os agentes precisam ter as autoridades e permissões adequadas, que em geral são verificados pelo servidor local, uma vez que é pressuposto ser este uma entidade confiável. Ou seja, mesmo a interação direta através da invocação de um método de outro agente é realizada através da chamada intermediária de um procedimento do servidor.

Objetos móveis Além de permitir que elementos ativos, como agentes, possam se movimentar entre lugares, alguns ambientes, como o Voyager[7] também dá suporte à movimentação de elementos passivos (objetos). Neste caso, agentes e lugares tem a capacidade de mover objetos, para os

[1]Diferentes ambientes usam nomes específicos para operações similares.

quais mantém uma referência virtual, que pode ser usada para acessar os seus métodos independentemente de sua localização.

Eventos Eventos representam outra forma de interação entre agentes (e lugares). Agentes e/ou lugares podem registrar interesse na notificação da ocorrência de qualquer evento sinalizado por qualquer outro elemento. Geralmente, a sinalização de eventos está acoplada à execução de um procedimento de um lugar ou agente. Enquanto em alguns sistemas como o Telescript, a sinalização de um evento é somente local (em um lugar), outros ambientes permitem também a sinalização de eventos remotos.

Serviço de Diretório Embora não seja um conceito específico e nem padrão em modelos de agentes móveis, está claro que a maioria de sistemas de agentes móveis requer um serviço de localização de agentes, para possibilitar a interação entre os mesmos.

Autoridade Além da identidade, cada agente e lugar tem uma autoridade que identifica unicamente o indivíduo ou a organização no mundo real que estes representam. Pode-se pensar na autoridade como sendo o equivalente ao userId de sistemas multi-usuário. Todas as permissões de acesso a procedimentos e/ou recursos de agentes e lugares são especificados em relação a autoridades. Na maioria dos sistemas é possível a qualquer elemento consultar a autoridade de outro elemento e é vetada a falsificação e a ocultação de autoridade.

Região É um conjunto de lugares com a mesma autoridade.

Permissões Na maioria dos sistemas existe a possibilidade de definir permissões diferenciadas de execução de procedimentos para os elementos detentores de diferentes autoridades. Permissões podem ser consultadas por qualquer elemento, porém só podem ser alteradas pela autoridade com permissão explícita de alteração. Exemplos de permissão poderiam ser que um agente está autorizado a entrar em um dado lugar, invocar um procedimento de outro agente ou criar novos agentes. Neste último exemplo, a permissão do agente criador geralmente se transfere automaticamente para o agente criado.

3 Características de ambientes de desenvolvimento de agentes

As principais características comuns à maioria dos ambientes de desenvolvimento de agentes são as seguintes:

- Todos os ambientes disponibilizam uma linguagem de programação de agentes, que na maioria das vezes é uma linguagem interpretada e extensível, e baseada em uma linguagem bem difundida, como Java ou Tcl. Uma das vantagens é a facilidade de adaptação para todas as arquiteturas para as quais existe um interpretador para a linguagem. Além disto, linguagens interpretadas são mais facilmente extensíveis do que linguagens compiladas, permitindo assim a incorporação de comandos específicos para o transporte e comunicação entre agentes.

- Todos disponibilizam um servidor de agentes que precisa executar em cada nó da rede potencialmente visitada por um agente. As principais tarefas destes servidores incluem uma autenticação de agentes, a re-ativação de agentes transferidos de outro servidor, o provimento de acesso (restrito) a recursos da máquina hospedeira (disco, memória, etc.) e dados locais, bem como a implementação de mecanismos de transporte e comunicação entre agentes locais e remotos. Em muitos ambientes este servidor também funciona como um intermediário em todas as interações entre agentes, verificando as permissões de acesso e controle entre agentes.

- A capacidade de serialização do estado de agentes é outra característica comum a todos os sistemas de agentes móveis. Ela permite que o estado de execução de um agente possa ser codificado para um fluxo (stream) de bytes e a partir do qual seja possível re-criar uma ou mais threads que continuem a execução exatamente a partir deste estado codificado. Enquanto a linguagem Java já provê tal facilidade (para objetos com uma única thread e que implementem as interfaces `Serializable` ou `Externalizable`) em alguns ambientes esta facilidade precisa ser incorporada na forma de primitivas da linguagem de programação dos agentes.
- A maioria dos ambientes se baseia em protocolos no nível de transporte ou superiores bem estabelecidos, como o TCP/IP, o X.25, http ou Email para realizar a comunicação entre os servidores. No entanto, em cima deste protocolo geralmente implementam uma camada extra que lida com autenticação, sincronização entre servidores e o empacotamento do estado e código de agentes para o transporte.
- Muitos ambientes já permitem iniciar e interagir com uma programa baseado em agentes móveis através de browsers WWW.
- A maioria dos ambientes permite uma definição dinâmica de itinerários, isto é, itinerários que são definidos pelas interações prévias com outros agentes e lugares visitados pelo agente.

4 Sistemas existentes

Nesta seção mostraremos dois ambientes de programação de agentes móveis, o AgentTcl e o Aglet Workbench, apresentando suas principais componentes e características.

4.1 AgentTcl

AgentTcl[5] (*www.cs.dartmouth.edu/~agent*) é um sistema de agentes móveis desenvolvido no Dartmouth College que permite a programação de agentes móveis em Tcl, Java e Scheme. O sistema provê uma grande série de facilidades para a movimentação, a comunicação e a depuração de agentes, além de um elaborado sistema de segurança. A principal contribuição do sistema, no entanto, é o seu modelo de estaleiro (*docking system*), que permite que agentes móveis aguardem a conexão de computadores móveis à rede para poderem se transferir para os mesmos.

4.1.1 Arquitetura

A arquitetura de AgentTcl (figura 2) essencialmente consiste de 4 camadas: a camada mais baixa é a interface com o mecanismo de transporte, que pode se TCP/IP ou correio eletrônico). A camada do servidor controla os agentes executando em sua máquina, autenticando agentes que chegam, reiniciando agentes em um ambiente de execução adequado, além de implementar os mecanismos básicos para a comunicação entre agentes e sua migração. Na camada acima do servidor estão os ambientes de execução para as linguagens de programação de agentes: os interpretadores Tcl e Scheme e a máquina virtual Java (Sun JDK 1.2). Na camada mais alta estão os próprios agentes móveis, sendo que alguns deles implementam serviços do sistema. Exemplos são os agentes estacionários que implementam um monitoramento da rede, um servidor de nomes para RPC e o próprio docking system. Assim sendo, o servidor AgentTcl provê somente a infra-estrutura básica para a execução e migração de agentes, sendo que todos os demais serviços são implementados por agentes estacionários e dedicados.

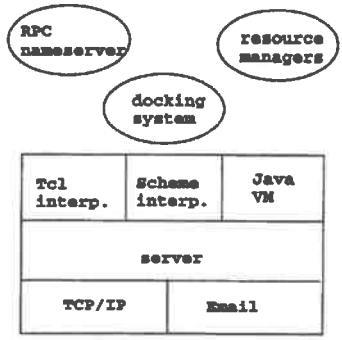

Figura 2: AgentTcl Architecture

4.2 A(s) linguagem(s) de programação

O projeto flexível de AgentTcl em princípio permite que qualquer linguagem interpretada possa ser usada para programar agentes, bastando para isto que se estenda o interpretador correspondente nos seguintes (a) inclusão de procedimentos para a codificação e decodificação do estado de agentes e (b) uma interface para o servidor de AgentTcl. A maior parte da interface entre os interpretadores e o servidor foi implementada em C/C++ e pode ser compartilhada entre todos os interpretadores, sendo que a componente específica a cada linguagem é apenas um conjunto de stubs que acessam esta interface.

A principal primitiva para a movimentação de agentes providas pelo servidor é `agent_jump $machine`, onde `$machine` é uma variável Tcl que contém o nome de uma máquina na rede. Através do acesso a atributos de um agente (p.ex. `$agent(home)`) é possível identificar a máquina onde o agente foi criado. Uma grande vantagem no uso de uma linguagem de scripts como o Tcl é que é possível se incorporar uma chamada a qualquer programa utilitário Unix dentro do código de um agente, aumentando assim enormemente a funcionalidade da linguagem de programação dos agentes.

Figura 3 mostra um trecho de código em AgentTcl para um agente que percorre um conjunto de máquinas descritas na variável `machineList`, executa o comando `who` do Unix e acrescenta o resultado deste comando em uma variável `output`, que depois é mostrada assim que o agente retorna para máquina de origem.

```
agent_begin
set output {}; set machineList {farofa feijoada torresmo}
foreach machine $machineList {
   agent_jump $machine
   append output [exec who]
}
agent_jump $agent(home)
output
agent_end
```

Figura 3: Código em AgentTcl

4.3 Agent RPC

Além do envio de mensagens e de byte streams, AgentTcl também provê a chamada remota de procedimentos (RPC) como mecanismo de comunicação de alto nível entre agentes, que podem estar localizados em uma única máquina ou em máquinas distintas da rede.

Como também ocorre em outros ambientes com RPC as interfaces de comunicação de um agente, isto é, seus procedimentos exportados, são especificados em uma Interface Definition Language (IDL), neste caso chamada de Agent IDL (AIDL), a partir da qual um compilador gera stubs usados pelos *agentes cliente* e *agente servidor* para realizar a comunicação. Além de tratar dos detalhes da transmissão e eventual codificação dos argumentos/resultados, o stub do servidor ainda realiza o registro da interface junto ao servidor de nomes *RPC nameserver*, que por sua vez é consultado pelos agentes cliente antes da primeira chamada a um procedimento remoto do agente servidor.

A principal vantagem do RPC em AgentTcl é sua flexibilidade e extensibilidade da conexão através da AIDL. Esta permite que parâmetros de procedimentos sejam identificados independentemente de sua ordem relativa na especificação de interface e tenham valores default. Além disto, o teste de compatibilidade entre interfaces no ato do estabelecimento da conexão é feito de acordo com os tipos de parâmetros e não somente através da comparação do nome e versão das interfaces.

Assim, um agente cliente pode ter uma conexão com qualquer agente servidor que disponibilize uma interface que contenha pelo menos os procedimentos e parâmetros definidos na AIDL do agente cliente. Além disto, um cliente pode manter conexões simultâneas com vários servidores que satisfaçam o critério de compatibilidade de interfaces mencionado acima. Tal flexibilidade é fundamental para o ambiente dinâmico e aberto de agentes móveis, no qual agentes e lugares eventualmente implementados por pessoas diferentes precisam ser capazes de interagir.

4.4 Docking System

O modelo de estaleiro (docking system) de AgentTcl trata do problema no qual a origem de um agente é um computador móvel (laptop) que só se conecta temporariamente a rede e além disto a cada vez com um endereço de rede diferente. Neste modelo, determina-se portanto para cada laptop uma estação fixa da rede que será o estaleiro (dock) dos agentes aguardando a re-conexão do laptop. Para tal, cada estação da rede executa um agente dedicado chamado *dock master*, que mantém uma lista dos agentes aguardando a transferência para o computador móvel correspondente. Para economizar recursos computacionais das estações estaleiro, os agentes em espera ficam armazenados em disco na forma inativa.

Quando um laptop se reconecta à rede, este se comunica com o dock master na estação dock designada originalmente, informando também o seu novo endereço de rede. A partir daí o dock master transfere todos os agentes que tem o laptop como destino para o mesmo.

Note-se que este modelo é bastante genérico, sendo também útil para o caso de agentes que desejam visitar laptops em sua peregrinação pela rede, no caso de que tanto a máquina origem como a máquina destino de um agente são laptops, ou no caso de uma eventual falha de partição temporária da rede. Para este último caso, no entanto, seria necessário definir uma estação dock para cada outra estação da rede. Ou seja, se um agente não conseguisse se transferir para uma estação X qualquer da rede, seria em vez disto transferido para uma outra estação Y que seja doc_X. Se Y também não estiver disponível, ficaria aguardando em doc_Y, e assim por diante.

4.4.1 Aplicações

Até agora, AgentTcl tem sido usado primordialmente para aplicações de busca de informação distribuída. Um exemplo concreto de uma aplicação é um sistema de agentes coletores de informações em

relatórios técnicos. Nesta aplicação os agentes móveis interagem com agentes dedicados à manutenção dos relatórios técnicos de diferentes instituições e destes coletam as identificações dos relatórios a partir da busca de palavras chave em seu texto. Além disto, agentes são replicados para efetuar a busca em paralelo e resultados parciais destes clones são coletados por um agente mestre, que se encarrega de transportar os resultados de volta para a estação origem e apresentá-los ao usuário.

4.5 IBM Aglet Workbench

Aglets Workbench (AWB)[6, 11] (*www.trl.ibm.co.jp/aglets*) é um ambiente para o desenvolvimento de agentes móveis baseado na linguagem Java e desenvolvido no Tokyo Research Laboratory da IBM. Atualmente a IBM está empenhada no criação de um padrão para agentes móveis baseado em algumas componentes e características do AWB, como por exemplo o Agent Transport Protocol e a J-AAPI.

Aglets (= applet + agent) são os agentes móveis em AWB que estendem o modelo de código carregável dinamicamente introduzido por Java applets e que como estes, executam no contexto de uma aplicação Java. Em AWB esta aplicação é o servidor de aglets, que precisa estar executando em todas as máquinas do sistema distribuído. Este servidor instancia o Java security manager para restringir e controlar as ações de aglets vindos de outros servidores. A transferência destes aglets é feita através de um carregador dinâmico de classes, o *network agent class loader*, que transfere os arquivos com a classe e o estado de um aglet de uma máquina remota para a máquina local.

AWB disponibiliza as classes `Aglet`, `AgletIdentifier`, `AgletProxy`, `Itinerary` e `Message` e as interfaces `AgletContext`, `FutureReply` e `MessageManager` como componentes da Java Aglet Application Programming Interface J-AAPI. Estas classes e interfaces abstratas servem de base para a programação de uma aplicação baseada em aglets.

As principais características do AWB podem ser resumidas na seguinte lista:

- manutenção de um espaço de nomes global, uniforme e imutável para aglets
- possibilidade de descrever itinerários complexos com vários destinos e com tratamento automático de falhas
- um mecanismo de comunicação *whiteboard* para a cooperação e o compartilhamento de informação entre aglets
- envio de mensagens síncrono e assíncrono entre agentes
- o conceito de contexto, que equivale ao conceito de lugar (vide seção 2), que provê uma interface padrão para a interação do aglet com o seu lugar, ou seja, para o acesso ao recursos locais da máquina em que está executando
- o conceito de proxy de um aglet, que representa o aglet e é um intermediário no acesso aos métodos públicos do aglet
- o modelo de callback para interações entre aglets e destes com o seu lugar
- um conjunto de padrões de uso de aglets (templates) para facilitar a programação de aglets
- possibilidade de disparar um conjunto de agentes a partir de um browser Web capaz de executar applets Java

4.5.1 As primitivas

Em AWB, as seguintes são as ações primitivas sobre aglets:

create um aglet é criado

clone um novo aglet é criado com código, estado e lugar idênticos aos de seu aglet criador

dispatch um aglet é transferido para outro destino

retract um aglet é transferido de volta para o lugar que chamou este método

deactivate um aglet é colocado em um estado inativo e seu estado é armazenado em um arquivo

activate a execução de um aglet inativo é re-iniciada a partir do seu último estado armazenado

dispose um aglet é removido do sistema

Para isto, existem métodos correspondentes, tanto na classe `Aglet` como na classe `AgletContext`, sendo que os métodos da primeira são usados quando um aglet executa as operações sobre si mesmo e as da segunda classe são usados para as operações entre aglets.

4.5.2 Contexto

Um *lugar* de execução de agentes no AWB é chamado de contexto e é definido pela interface `AgletContext`. Esta interface provê um conjunto padronizado de serviços disponíveis para os aglets, tais como criação de outros aglets, transferência para outro contexto, clonagem, etc.

Para executar qualquer operação de controle, primeiro o aglet precisa obter o acesso ao seu contexto, o que é feito através do método `getAgletContext`. Em seguida pode por exemplo criar um novo aglet chamando o método `createAglet(getCodeBase(), ``MyAglet'', null)`, onde o primeiro argumento é uma URL que descreve um diretório (remoto) que contém a classe compilada, o segundo é o nome do arquivo contendo a classe e o terceiro são argumentos de criação.

Portanto é o método do contexto o responsável pela obtenção da classe desejada, a criação de uma thread de execução e a inicialização do novo agente, através da chamada do construtor `Aglet`. Na verdade, a inicialização de um aglet como também todas as demais operações sobre aglets geralmente requer a participação do próprio aglet sendo manipulado. Isto é realizado através do modelo callback.

4.5.3 O Modelo Callback

Para cada uma das possíveis ações, digamos `AçãoX`, sobre um aglet existe também um método callback chamado `onAçãoX`, que permite ao aglet reagir a (ou se preparar para) a ação que está sendo executada sobre ele. Por exemplo, é comum se incluir código de inicialização no método `onCreation`, que é chamada logo após o retorno do método construtor `Aglet` e antes da chamada do método `run`, que efetivamente inicia a execução do aglet.

Ao contrário deste exemplo, também existem métodos callback que são executados *antes* do início da ação. Exemplos são `onCloning` e `onDispatching`. O primeiro método geralmente determina se a clonagem do aglet está autorizada ou se ela deve gerar uma exceção `SecurityException`. O segundo método permite que um aglet codifique o seu estado de execução antes da transferência para outra máquina. Por outro lado, após a transferência do aglet, é executado o método `onArrival`, para inicializar o aglet antes de sua ativação no novo lugar.

O princípio descrito acima é chamado de *modelo callback* e é essencial para controlar se as ações sobre os agentes estão autorizadas e para permitir que o próprio agente se prepare para a ação. A figura 4 mostra um esboço das chamadas antes e depois da transferência de um aglet.

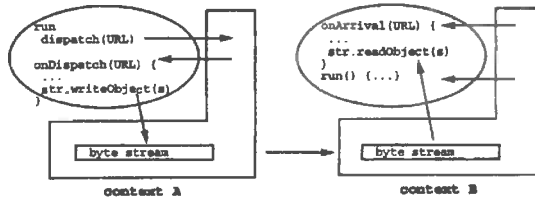

Figura 4: O modelo callback

4.5.4 Proxy de Aglets

Para evitar que qualquer aglet tenha acesso direto aos métodos públicos de um outro aglet, criou-se o conceito de proxy. Um proxy de um aglet funciona como o intermediário entre o aglet solicitante da ação e o aglet alvo da ação. Um proxy é criado para cada aglet e uma referência para o proxy é retornada na chamada de criação de um novo aglet, isto é:

```
AgletProxy proxy = context.createAglet(...)
```

De posse de um objeto proxy, é possível então executar todas as ações sobre o aglet representado pelo proxy, como por exemplo, mandar o aglet para um outro lugar `proxy.dispatch(URL)`), ou enviar uma mensagem para o aglet, `proxy.sendAsynchMessage(msg)`.

As principais funções do proxy são de (a) interceptar qualquer chamada de um método em um aglet e verificar se o elemento solicitante está credenciado a executar a ação e (b) redirecionar chamadas a aglets para a sua localização corrente. Através desta segunda função, é possível programar uma aplicação baseada em agentes sem que o programador precise se preocupar com a localização corrente do agente alvo da ação.

4.5.5 Envio de Mensagens

No AWB é possível mandar mensagens síncronas ou assíncronas entre agentes. Uma mensagem é um objeto da classe **Message** e essencialmente contém três atributos: o seu tipo, que é um string que caracteriza uma classe de mensagens, o argumento que pode ser um valor atômico (inteiro, booleano, etc.) ou uma tabela de argumentos, e um timestamp.

O envio de mensagens ocorre através do proxy dos aglets, que funciona como o roteador (*gateway*) para o aglet e permite que se efetue o envio da mensagem com total transparência de localização, ou seja, sem se conhecer a localização do aglet destinatário. Para isto, basta que se chame o método **sendMessage** ou **sendAsyncMessage** no proxy do aglet.

Quando se usa o envio assíncrono de mensagens, o método **sendAsyncMessage(msg)** retorna um objeto da classe **FutureReply**, que serve de handle para que o remetente de **msg**, posteriormente possa verificar se recebeu uma resposta do aglet destinatário de msg.

O recebimento da mensagem por um aglet segue o modelo callback, ou seja, ao chegar uma mensagem para um aglet, o contexto local ao aglet destinatário chama o seu método **handleMessage** que deve ser alterado pelo programador do aglet para tratar o recebimento da mensagem. Este método retorna o valor booleano **true** caso a mensagem tenha sido tratada e **false** se foi tratada ou se causou uma exceção. Em geral, o método contém um case que executa diferentes funções dependendo do tipo da mensagem recebida.

4.5.6 O Agent Transfer Protocol

O Agent Transport Protocol (ATP) é usado pelos servidores de aglets para transferir o código e estado de aglets de uma máquina para outra. O ATP é um protocolo padrão (no nível de aplicação) para a Internet, que usa os Universal Resource Locators (URLs) como forma de endereçar recursos em sistemas de agentes. O ATP também disponibiliza um mecanismo uniforme para transferência de agentes e funções padrão para a consulta remota sobre agentes.

Uma primeira versão de ATP foi implementada em Java como um pacote auto-contido do AWB que pode ser usado também em outros ambientes de programação de agentes móveis. Este pacote consiste de um conjunto de classes que definem uma API para a criação de ATP daemons, conexão entre estes e geração de solicitações respostas no padrão ATP.

4.6 Outros Sistemas

Além dos ambientes mencionados, existe um grande número de outros sistemas comerciais e de domínio público para o desenvolvimento de aplicações baseadas em agentes móveis. Segue uma lista dos projetos mais conhecidos com as respectivas *homepages*.

ARA	Desenvolvimento de uma infra-estrutura portável e segura para execução de agentes móveis em redes heterogêneas. Universität Kaiserslautern, *www.uni-kl.de/AG-Nehmer/Ara/index.html*
Coast	Um sistema de agentes móveis para a detecção de intrusos, vírus e ataques à segurança em redes. Purdue University, *www.cs.purdue/homes/mcrosbie/research*
DIAG	Pesquisa nos aspectos de coordenação e planejamento de sistemas de agentes móveis autônomos. University of Michigan, *ai.eecs.umich.edu/diag/homepage.html*
InteRRap	Arquitetura em camadas para o desenvolvimento de agentes autônomos e cooperativos. Universität Saarbrücken, *www.dfki.uni-sb.de/ jpm/interrap.html*
MOA	Ambiente genérico para a programação de agentes móveis em Java. Open Group, *www.opengroup.org/RI/java/moa/index.html*
Procura Project	Sistema de agentes móveis para a gerência automática de redes de computadores com capacidade de auto-configuração. Carleton University, CA, *www.sce.carleton.ca/netmanage/perpetum.shtml*
Tacoma	Desenvolvimento de um sistema operacional para agentes móveis e investigação de como agentes móveis podem realizar funções tradicionais de sistemas operacionais. Cornell e Tromso's University *www.cs.uit.no/DOS/Tacoma*
Voyager	Ambiente de programação de agentes móveis comercial baseado em Java e com um object request broker compatível com CORBA. ObjectSpace[7] e *www.objectspace.com/voyager/voyager2.html*

5 Conclusão

Pesquisa e desenvolvimento em ambientes de suporte e aplicações para agentes móveis está passando por um momento de intensa atividade, mas a maioria dos pesquisadores nesta área concordam em que a utilidade e a aplicabilidade deste novo paradigma estão longe de ter sido esgotadas. O mais importante, porém, é que agentes móveis estão mudando radicalmente a natureza e a arquitetura de aplicações

distribuídas, proporcionando novas e revolucionárias facilidades.

O futuro desenvolvimento nesta área provavelmente deverá ser fortemente influenciado pelas atuais e futuras aplicações distribuídas que melhor se adequarem ao paradigma de agentes móveis.

Por fim, gostaríamos de discutir brevemente os assuntos que acreditamos tornar-se-ão objeto de futura pesquisa e desenvolvimento nesta área.

Segurança Este assunto certamente será o mais pesquisado nos próximos anos no contexto de agentes móveis. Vários problemas de autenticação, controle de permissões e segurança de comunicação em geral ainda precisam ser resolvidos para tornar agentes móveis uma tecnologia utilizável em larga escala. Em [3] é apresentado um modelo de segurança para agentes, onde os autores classificam os tipos de ataques em tais sistemas. São eles i) monitoramento de um canal de comunicação; ii) interceptação e troca de código ou dados durante a transferência; iii) cópia de um agente interceptado e re-transmissão fora de seu propósito original; iv) uso exaustivo de recursos do nó hospedeiro de forma que todos os demais agentes ficam impossibilitados de executar; e v) interceptação na interação entre agentes. Além do perigo de ataques devido ao carregamento dinâmico de código remoto (como em applets), agentes móveis precisam estar também protegidos contra ataques vindos do ambiente de execução (o servidor) nas máquinas da rede. Isto requer um modelo complexo de segurança, com mecanismos e regras que permitam implementar formas da autenticação, criptografia e controle de acesso para domínios e autoridades em tais sistemas.

Tolerância a Falha Em aplicações que necessitam um alto grau de confiabilidade e disponibilidade de seus agentes, a mobilidade de agentes traz consigo novos problemas. Estes também são derivados da dependência de agentes com relação ao ambiente de execução. Em particular, precisa-se pensar em maneiras de evitar que os dados transportados por um agente sejam perdidos quando o mesmo é destruído, por exemplo devido ao desligamento ou falha do nó em que estava executando.

Outro problema é o decorrente da possível falha no meio de comunicação. Neste caso, um agente poderia ficar "ilhado" em um lugar sem poder ser transferido para o próximo lugar segundo o seu itinerário. Soluções parciais para este problema tem sido propostas, como por exemplo o modelo de estaleiro (docking system) em AgentTcl, mas este assume que a falha (ou ausência do canal) de comunicação seja temporária e que os agentes não tenham requisitos de sincronização, como por exemplo de encontro com outros agentes. Para se tentar solucionar problemas mais gerais de falha de comunicação, precisaria-se introduzir a noção de itinerários alternativos que pudessem ser escolhidos dinamicamente pelo servidor ou pelo próprio agente.

Agentes cooperantes e coordenação Outro assunto de pesquisa bastante relevante para agentes móveis é a teoria de coordenação e planejamento para elementos autônomos e linguagens para a sua especificação. Pesquisa nesta área é inter-disciplinar e tem sido feita por pesquisadores em inteligência artificial, pesquisa operacional e teoria da organização. Estes estudam formas de representação e de manipulação de conhecimento coletivo, bem como mecanismos e linguagens para a implementar sincronização, planejamento e cooperação entre agentes. Este assunto é relevante na medida em que aplicações distribuídas sejam compostas de agentes que precisam coordenar as suas ações a fim de alcançar um objetivo comum. Para isto, será necessário especificar planos ou estratégias globais que devem determinar a ação e/ou movimentação de cada agente. Além dos problemas referentes à coordenação descentralizada, em agentes móveis também seria necessário tratar de como os agentes móveis teriam acesso e/ou manipulariam estes planos de cooperação.

Representação de Conhecimento O quanto mais complexa for a tarefa a ser executada por um conjunto de agentes, tanto mais elaborada também deverão ser as formas de representação e inferência de conhecimento da aplicação. Este assunto têm sido tradicionalmente pesquisado

pela comunidade trabalhando em inteligência artificial. Com agentes móveis, no entanto, surgem novos desafios no que diz respeito à modelagem de conhecimento social e a formalização da semântica de interações entre agentes. Assim, acredito que deverão surgir cada vez mais projetos interdisciplinares nesta área, que terão uma uma forte participação da comunidade de inteligência artificial.

Existem também algumas vozes críticas que afirmam que agentes móveis não passam de mais uma tecnologia que simplesmente está em moda e que como outras tecnologias, por exemplo a própria inteligência artificial, ainda não convenceu de que trará benefícios substanciais à forma como atualmente usamos computadores em rede. Como mencionado anteriormente, acreditamos também que o sucesso desta tecnologia, ou seja, o seu emprego em larga escala, vai depender fortemente da existência de aplicações comerciais importantes que façam uso da mesma.

Independente desta incerteza quanto ao futuro desta tecnologia, agentes móveis por sí só já constituem um assunto importante e interessante, que tem como a sua maior contribuição a proposta de um novo paradigma para a modelagem e o desenvolvimento de aplicações distribuídas.

Referências

[1] Krishna Bharat and Luca Cardalli. Migratory applications. In *Proc. of ACM Symposium on User Interface Software and Technology '95,*, November 1995.

[2] Fah-Chun Cheong. *Internet Agents: Spiders, Wanderers, Brokers, and Bots.* New Riders, 1996.

[3] G. Karjoth, D.B. Lange, and M. Oshima. A Security Model fo Aglets. *IEEE Internet Comuting*, 1(4):68–77, July 1997.

[4] J. Kiniry and D. Zimmerman. A Hands-on Look at Java Mobile Agents. *IEEE Internet Comuting*, 1(4):21–30, July 1997.

[5] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. Agent TCL: Targeting the Needs of Mobile Computers. *IEEE Internet Comuting*, 1(4):58–66, July 1997.

[6] D. Lange and M. Oshima. *Programming Mobile Agents in Java (TM) - With the Java Aglet API.* 1998. pre-print version in ttp://www.trl.ibm.co.jp/aglets/aglet-book/index.html.

[7] ObjectSpace. The Agent ORB for Java. White paper, ObjectSpace, Inc, 14881 Quorum Dr., Suite 400, Dallas TX 75240, September 1997. www.objectspace.com/Voyager/.

[8] J. Ousterhout. *Tcl and the Tk Toolkit.* Professional Computing Series. Addison Wesley, 1994.

[9] K. Rothermel and R. Popescu-Zeletin. *Mobile Agents.* Number 1219 in LNCS. Springer Verlag, 1997.

[10] G. Susilo. Infrastructure for advanced network management based on mobile code. Technical Report SCE-97-10, Systems and Computer Engineering, Carleton University, June 1997.

[11] B. Venners. Under the Hood: The Architecture of Aglets. *JavaWorld*, April 1997. www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html.

[12] J. White. Mobile agents white paper. White paper, General Magic, 1997. www.genmagic.com/agents/Whitepaper/whitepaper.html.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1995 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

Thomaz I. Seidman e Carlos Humes Jr.
SOME KANBAN-CONTROLLED MANUFACTURING SYSTEMS: A FIRST STABILITY ANALYSIS
RT-MAC-9501, janeiro de 1995, 19 pp.

C.Humes Jr. and A.F.P.C. Humes
STABILIZATION IN FMS BY QUASI- PERIODIC POLICIES
RT-MAC-9502, março de 1995, 31 pp.

Fabio Kon e Arnaldo Mandel
SODA: A LEASE-BASED CONSISTENT DISTRIBUTED FILE SYSTEM
RT-MAC-9503, março de 1995, 18 pp.

Junior Barrera, Nina Sumiko Tomita, Flávio Soares C. Silva, Routo Terada
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY PAC LEARNING
RT-MAC-9504, abril de 1995,16 pp.

Flávio S. Corrêa da Silva e Fabio Kon
CATEGORIAL GRAMMAR AND HARMONIC ANALYSIS
RT-MAC-9505, junho de 1995,17 pp.

Henrique Mongelli e Routo Terada
ALGORITMOS PARALELOS PARA SOLUÇÃO DE SISTEMAS LINEARES
RT-MAC-9506, junho de 1995, 158 pp.

Kunio Okuda
PARALELIZAÇÃO DE LAÇOS UNIFORMES POR REDUÇÃO DE DEPENDÊNCIA
RT-MAC-9507, julho de 1995, 27 pp.

Valdemar W. Setzer e Lowell Monke
COMPUTERS IN EDUCATION: WHY, WHEN, HOW
RT-MAC-9508, julho de 1995, 21 pp.

Flávio S. Corrêa da Silva
REASONING WITH LOCAL AND GLOBAL INCONSISTENCIES
RT-MAC-9509, julho de 1995, 16 pp.

Julio M. Stern
MODELOS MATEMÁTICOS PARA FORMAÇÃO DE PORTFÓLIOS
RT-MAC-9510, julho de 1995, 43 pp.

Fernando Iazzetta e Fabio Kon
A DETAILED DESCRIPTION OF MAXANNEALING
RT-MAC-9511, agosto de 1995, 22 pp.

Flávio Keidi Miyazawa e Yoshiko Wakabayashi
POLYNOMIAL APPROXIMATION ALGORITHMS FOR THE ORTHOGONAL Z-ORIENTED 3-D PACKING PROBLEM
RT-MAC-9512, agosto de 1995, pp.

Junior Barrera e Guillermo Pablo Salas
SET OPERATIONS ON COLLECTIONS OF CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMINIG OF MORPHOLOGICAL MACHINES
RT-MAC-9513, agosto de 1995, 84 pp.

Marco Dimas Gubitoso e Jörg Cordsen
PERFORMANCE CONSIDERATIONS IN VOTE FOR PEACE
RT-MAC-9514, novembro de 1995, 18pp.

Carlos Eduardo Ferreira e Yoshiko Wakabayashi
ANAIS DA I OFICINA NACIONAL EM PROBLEMAS COMBINATÓRIOS: TEORIA, ALGORITMOS E APLICAÇÕES
RT-MAC-9515, novembro de 1995, 45 pp.

Markus Endler and Anil D'Souza
SUPPORTING DISTRIBUTED APPLICATION MANAGEMENT IN SAMPA
RT-MAC-9516, novembro de 1995, 22 pp.

Junior Barrera, Routo Terada,
Flávio Corrêa da Silva and Nina Sumiko Tomita
AUTOMATIC PROGRAMMING OF MMACH'S FOR OCR
RT-MAC-9517, dezembro de 1995, 14 pp.

Junior Barrera, Guillermo Pablo Salas and Ronaldo Fumio Hashimoto
SET OPERATIONS ON CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMING OF MMACH'S
RT-MAC-9518, dezembro de 1995, 14 pp.

Daniela V. Carbogim and Flávio S. Corrêa da Silva
FACTS, ANNOTATIONS, ARGUMENTS AND REASONING
RT-MAC-9601, janeiro de 1996, 22 pp.

Kunio Okuda
REDUÇÃO DE DEPENDÊNCIA PARCIAL E REDUÇÃO DE DEPENDÊNCIA GENERALIZADA
RT-MAC-9602, fevereiro de 1996, 20 pp.

Junior Barrera, Edward R. Dougherty and Nina Sumiko Tomita
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY DESIGN OF STATISTICALLY OPTIMAL OPERATORS IN THE CONTEXT OF COMPUTATIONAL LEARNING THEORY.
RT-MAC-9603, abril de 1996, 48 pp.

Junior Barrera e Guillermo Pablo Salas
SET OPERATIONS ON CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMINIG OF MMACH'S
RT-MAC-9604, abril de 1995, 66 pp.

Kunio Okuda
CYCLE SHRINKING BY DEPENDENCE REDUCTION
RT-MAC-9605, maio de 1996, 25 pp.

Julio Stern, Fabio Nakano e Marcelo Lauretto
REAL: REAL ATTRIBUTE LEARNING FOR STRATEGIC MARKET OPERATION
RT-MAC-9606, agosto de 1996, 16 pp.

Markus Endler
SISTEMAS OPERACIONAIS DISTRIBUÍDOS: CONCEITOS, EXEMPLOS E TENDÊNCIAS
RT-MAC-9607, agosto de 1996, 120 pp.

Hae Yong Kim
CONSTRUÇÃO RÁPIDA E AUTOMÁTICA DE OPERADORES MORFOLÓGICOS E EFICIENTES PELA APRENDIZAGEM COMPUTACIONAL
RT-MAC-9608, outubro de 1996, 19 pp.

Marcelo Finger
NOTES ON COMPLEX COMBINATORS AND STRUCTURALLY-FREE THEOREM PROVING
RT-MAC-9609, dezembro 1996, 28 pp.

Carlos Eduardo Ferreira, Flávio Keidi Miyazawa e Yoshiko Wakabayashi (eds)
ANAIS DA I OFICINA NACIONAL EM PROBLEMAS DE CORTE E EMPACOTAMENTO
RT-MAC-9610, dezembro de 1996, 65 pp.

Carlos Eduardo Ferreira, C. C. de Souza e Yoshiko Wakabayashi
REARRANGEMENT OF DNA FRAGMENTS: A BRANCH-AND-CUT ALGORITHM
RT-MAC-9701, janeiro de 1997, 24 pp.

Marcelo Finger
NOTES ON THE LOGICAL RECONSTRUCTION OF TEMPORAL DATABASES
RT-MAC-9702, março de 1997, 36 pp.

Flávio S. Corrêa da Silva, Wamberto W. Vasconcelos e David Robertson
COOPERATION BETWEEN KNOWLEDGE BASED SYSTEMS
RT-MAC-9703, abril de 1997, 18 pp.

Junior Barrera, Gerald Jean Francis Banon, Roberto de Alencar Lotufo, Roberto Hirata Junior
MMACH: A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9704, maio de 1997, 67 pp.

Julio Michael Stern e Cibele Dunder
PORTFÓLIOS EFICIENTES INCLUINDO OPÇÕES
RT-MAC-9705, maio de 1997, 29 pp.

Junior Barrera e Ronaldo Fumio Hashimoto
COMPACT REPRESENTATION OF W-OPERATORS
RT-MAC-9706, julho de 1997, 13 pp.

Dilma M. Silva e Markus Endler
CONFIGURAÇÃO DINÂMICA DE SISTEMAS
RT-MAC-9707, agosto de 1997, 35 pp

Kenji Koyama e Routo Terada
AN AUGMENTED FAMILY OF CRYPTOGRAPHIC PARITY CIRCUITS
RT-MAC-9708, setembro de 1997, 15 pp

Routo Terada e Jorge Nakahara Jr.
LINEAR AND DIFFERENTIAL CRYPTANALYSIS OF FEAL-N WITH SWAPPING
RT-MAC-9709, setembro de 1997, 16 pp

Flávio S. Corrêa da Silva e Yara M. Michelacci
MAKING OF AN INTELLIGENT TUTORING SYSTEM (OR METHODOLOGICAL ISSUES OF ARTIFICIAL INTELLIGENCE RESEARCH BY EXAMPLE)
RT-MAC-9710, outubro de 1997, 16 pp.

Marcelo Finger
COMPUTING LIST COMBINATOR SOLUTIONS FOR STRUCTURAL EQUATIONS
RT-MAC-9711, outubro de 1997, 22 pp.

Maria Angela Gurgel and E.M.Rodrigues
THE F-FACTOR PROBLEM
RT-MAC-9712, dezembro de 1997, 22 pp.

Perry R. James, Markus Endler, Marie-Claude Gaudel
DEVELOPMENT OF AN ATOMIC-BROADCAST PROTOCOL USING LOTOS
RT-MAC-9713, dezembro de 1997, 27 pp.

Carlos Eduardo Ferreira and Marko Loparic
A BRANCH-AND-CUT ALGORITHM FOR A VEHICLE ROUTING PROBLEM WITH CAPACITY AND TIME CONSTRAINTS
RT-MAC-9714, dezembro de 1997, 20 pp.

Nami Kobayashi
A HIERARCHY FOR THE RECOGNIZABLE M-SUBSETS
RT-MAC-9715, dezembro de 1997, 47 pp.

Flávio Soares Corrêa da Silva e Daniela Vasconcelos Carbogim
A TWO-SORTED INTERPRETATION FOR ANNOTATED LOGIC
RT-MAC-9801, fevereiro de 1998, 17 pp.

Flávio Soares Corrêa da Silva, Wamberto Weber Vasconcelos, Jaume Agustí, David Robertson e Ana Cristina V. de Melo.
WHY ONTOLOGIES ARE NOT ENOUGH FOR KNOWLEDGE SHARING
RT-MAC-9802, outubro de 1998, 15 pp.

J. C.de Pina e J. Soares
ON THE INTEGER CONE OF THE BASES OF A MATROID
RT-MAC-9803, novembro de 1998, 16 pp.

K. Okuda and S.W.Song
REVISITING HAMILTONIAM DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9804, dezembro 1998, 17pp.

Markus Endler
AGENTES MÓVEIS: UM TUTORIAL
RT-MAC-9805, dezembro 1998, 19pp.