Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA

Francisco J. S. Vasconcellos

Centro de Instrução Almirante Wandenkolk
Marinha do Brasil
Rio de Janeiro, RJ - CEP 20091-000

Edmundo R. M. Madeira

IC - Instituto de Computação
UNICAMP - Universidade Estadual de Campinas
Campinas, SP - CEP 13081-970

E-mail: {fvascon, edmundo}@dcc.unicamp.br

Resumo

Sistemas de Agentes Móveis têm recebido muita atenção ultimamente. Os Agentes dotados de mobilidade, representando remotamente aplicações e usuários, propiciam vantagens para certas aplicações como a redução do tráfego na rede, maior flexibilidade, possibilidade de balanceamento de carga e processamento assíncrono, permitindo assim o desenvolvimento de soluções mais eficientes para problemas relacionados a comércio eletrônico, computação móvel, gerência remota de recursos, *data mining*, entre outros.

Este artigo descreve a modelagem e implementação de um suporte a Agentes Móveis sobre um *broker* que segue a especificação CORBA, estabelecendo as vantagens de se aliar as funcionalidades oferecidas pelo *broker* com a capacidade de migração de objetos.

Palavras-chave: Processamento Distribuído Aberto, Agentes Móveis, Sistemas de Agentes, Facilidade de Agentes Móveis, CORBA.

Abstract

Mobile Agents Systems have received much attention lately. Agents with mobility, representing remotely applications and end-users, provide advantages as: network traffic reduction, flexibility, load balancing and asynchronous processing. This paradigm allows the development of more efficient solutions to problems related to areas like electronic market, mobile computing, remote management of resources, data mining, among others.

This paper describes the modeling and implementation of a Mobile Agent Support over a *broker* CORBA-compliant, establishing the advantages to associate the ORB functionalities with the object migration capability.

Keywords : Open Distributed Processing, Mobile Agents, Agents Systems, Mobile Agents Facility, CORBA.

1 Introdução

O rápido avanço da tecnologia vem possibilitando a construção de sistemas de computação compostos por um grande número de processadores ligados através de redes de alta velocidade. Estes sistemas distribuídos apresentam inúmeras vantagens sobre sistemas centralizados como escalabilidade, confiabilidade, desempenho, entre outros. O processamento distribuído aberto, que procura englobar a heterogeneidade de meios, proporcionando a interoperabilidade e portabilidade de aplicações entre sistemas geograficamente separados, vem sendo estudado e padronizado pela ISO e ITU-T através de seu Modelo de Referência para Processamento Distribuído Aberto e encontra, no consórcio do OMG (*Object Management Group*), um aliado na tentativa de desenvolvimento deste padrão.

Entre as diversas abordagens da distribuição do processamento, uma que vem despertando cada vez mais interesse é a possibilidade de se migrar o código para execução remota de uma aplicação. Um Agente Móvel é um programa que pode migrar, levando consigo seu código e estado de execução, de um *host* para outros, de modo autônomo, a fim de realizar uma tarefa em benefício de uma determinada aplicação ou usuário que o lançou.

Muito se tem estudado e escrito, especialmente relacionado a aplicações para a Internet, sobre os Agentes Móveis e sobre a infra-estrutura necessária a estes Agentes. O OMG, entidade criadora da especificação CORBA (*Common Object Request Broker Architecture*), reconhecendo os benefícios deste paradigma, vem efetuando esforços no sentido de especificar a Facilidade de Agentes Móveis para sua Arquitetura de Gerência de Objetos - OMA (*Object Management Architecture*). Porém, nota-se que desde a emissão do *Request for Proposal* para esta finalidade, muito já se fugiu da proposta inicial.

Este trabalho, de modo diverso aos demais relacionados e ao atual enfoque dado pelo OMG, visa a aplicação dos Agentes Móveis em um ambiente distribuído aberto, que pode ser uma WAN ou LAN estando ou não esta ligada a Internet, e demonstra a possibilidade de se utilizar um *broker* para migrar Agentes, aliando-se assim às vantagens oferecidas pela CORBA, os benefícios advindos da possibilidade de se mover o código e estado de objetos em execução.

Este artigo apresenta na Seção 2 um breve resumo sobre a Plataforma Multiware, desenvolvida na UNICAMP, que utiliza como middleware um *broker* que segue a especificação do OMG. Descreve ainda os conceitos relacionados aos Agentes Móveis, suas vantagens e restrições, bem como apresenta o estado atual do esforço de especificação da Facilidade de Agentes Móveis do OMG. Na Seção 3 é apresentada a modelagem do Suporte a Agentes móveis para a Multiware, sendo descritas as funcionalidades esperadas deste suporte. Na Seção 4, são discutidos os detalhes de implementação, demonstrando-se como se chegou às funcionalidades requeridas e, na Seção 5, são apresentados os trabalhos relacionados, estabelecendo-se as diferenças desta proposta. Finalmente na Seção 6, é apresentada a conclusão do trabalho.

2 Conceitos iniciais

Nesta seção apresentamos a plataforma Multiware, os conceitos relacionados a CORBA e aos agentes móveis e, por fim, como o OMG vem trabalhando no sentido de especificar a Facilidade de Agentes Móveis para a sua Arquitetura.

2.1 O OMG e sua Arquitetura de Gerência de Objetos

O *Object Management Group* foi formado em 1989 com o propósito de desenvolver padrões que admitissem a interoperabilidade e portabilidade de aplicações distribuídas orientadas a objeto. O OMG produz especificações utilizando idéias e tecnologias de seus membros que respondem aos *Requests For Information* (RFI) e *Requests For Proposals* (RFP) emitidos. Entre os membros, quase 800, estão algumas das maiores empresas comerciais da área de computação distribuída orientada a objeto (IBM, HP, Digital, Iona, Tandem, entre outras) e instituições de pesquisas, o que torna forte esta abordagem. A Arquitetura de Gerência de Objetos (*Object Management Architecture* - OMA), mostrada na Figura 1, estabelece uma estrutura com termos e definições para especificações de interfaces de objetos em um ambiente distribuído.

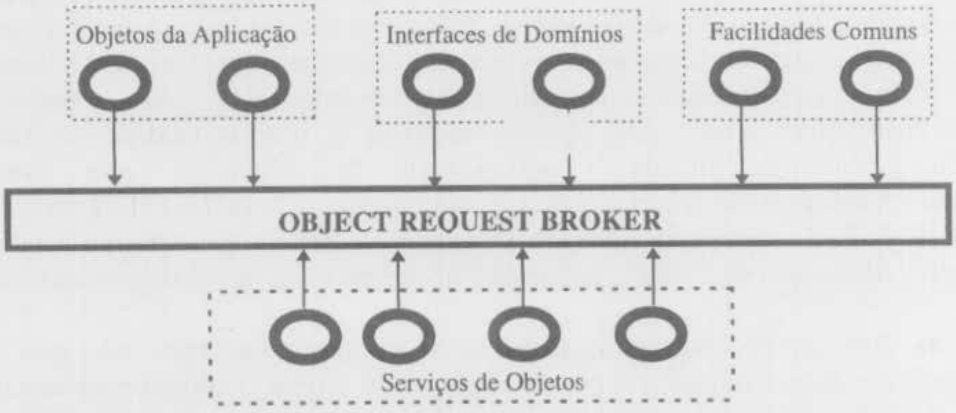

Figura 1 - Arquitetura de Gerência de Objetos do OMG.

O **Negociador de Requisições de Objetos** ou *Object Request Broker* (ORB) define o meio que permite a interação entre objetos do ambiente distribuído. Atualmente temos a versão 2.2 da especificação CORBA [1].

Os **Serviços de Objetos** (*CORBAServices*) definem uma coleção de serviços (interfaces e objetos) que oferecem funções básicas para uso e implementação de objetos como, por exemplo, os serviços de tempo, transações e controle de eventos.

As **Facilidades Comuns** (*CORBAFacilities*) definem serviços com funções que algumas aplicações podem utilizar, mas que não são fundamentais como os CORBAServices. Neste conjunto temos a gerência de tarefas onde se encontra a Facilidade de Agentes Móveis.

As **Interfaces de Domínios** (*Domain Interfaces*) definem serviços para áreas específicas como telecomunicações, manufaturas e outras. Até 1997, estes serviços eram definidos na Arquitetura dentro das *CORBAFacilities* com a denominação de Facilidades Verticais.

Os **Objetos de Aplicação** (*Application Objects*) definem as aplicações e serviços que não são padronizados pelo OMG porque as funções são associadas às necessidades específicas das aplicações.

Os componentes são descritos em uma linguagem puramente declarativa chamada IDL (***Interface Definition Language***) que define as interfaces dos objetos. Assim estes se tornam portáveis através de linguagens, ferramentas, sistemas operacionais e redes de computadores. A linguagem IDL não provê detalhes de implementação. Pode-se usar IDL para definir APIs (*Application Program Interfaces*) e métodos de invocação para qualquer linguagem suportada pela CORBA (atualmente C, C++ , Smalltalk, Ada, COBOL e Java). Isto possibilita objetos cliente e servidor, escritos em linguagens diferentes, interoperarem através de redes de computadores e sistemas operacionais.

2.1.1 A CORBA (Common Object Request *Broker* Architecture)

A plataforma descrita pela CORBA é composta pelo núcleo do ORB e de suas interfaces de chamadas, permitindo a interação transparente entre os clientes e as implementações de objetos, como são conhecidos os provedores de serviços.

Denominamos **Objeto CORBA** um objeto ou conjunto de objetos que implementam uma interface definida em IDL e que, através de seu registro no ORB, disponibilizará uma referência de objeto para o uso de clientes de modo independente da linguagem de implementação, da plataforma de execução e de sua localização.

O **CLIENTE** é qualquer código que invoca um método em um Objeto CORBA, podendo inclusive ser outro Objeto CORBA.

Maiores detalhes podem ser obtidos em [1] que se encontra disponível em http://www.omg.org.

2.1.2 A MULTIWARE

A plataforma Multiware [2], ilustrada na Figura 2, possui três camadas básicas: Hardware/Software, Middleware e Groupware.

A camada de Hardware/Software representa o nível de abstração ligado aos Sistemas Operacionais, aos protocolos e equipamentos.

A camada Middleware é composta por ORBs como base e por novas funções de processamento distribuído aberto desenvolvidas sobre os ORBs.

Por fim, a camada Groupware é definida de modo a prover serviços para suporte a aplicações CSCW.

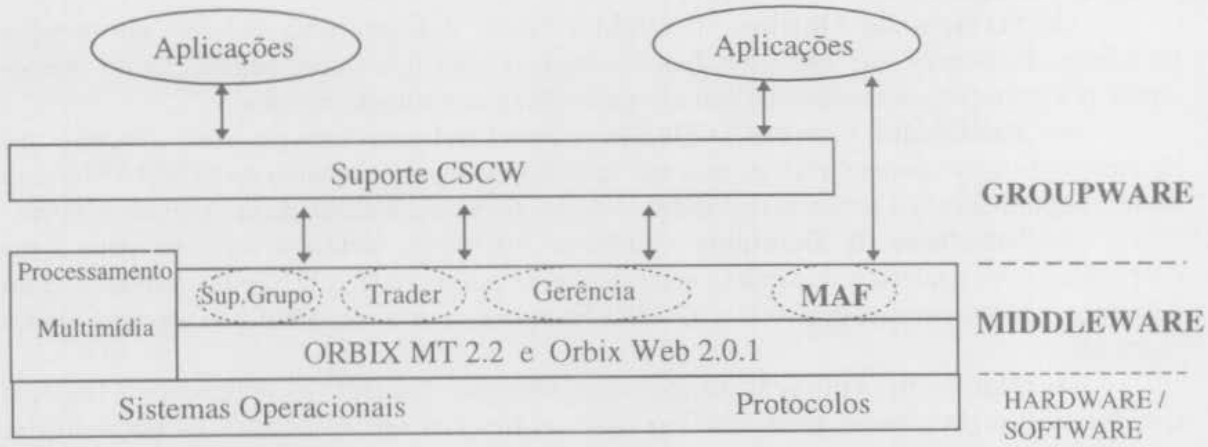

Figura 2 - Plataforma Multiware.

No projeto da Multiware, o Suporte a Agentes Móveis (MAF) se apresenta como uma nova função ODP, na camada Middleware, devido a diversidade de aplicações que podem fazer uso deste serviço, mas pode-se afirmar que também poderia fazer parte da camada Groupware já que provê funcionalidade para suporte a trabalho cooperativo.

2.2 Agentes e agentes móveis

O termo agentes é utilizado por diferentes áreas, como sistemas distribuídos, engenharia de software, inteligência artificial entre outras, possuindo portanto diferentes definições.

Mesmo em uma área específica, como IA (Inteligência Artificial) por exemplo, há controvérsias. Em [3] encontramos um apanhado de definições e uma tentativa de se classificar e definir agentes. Esta tentativa foi comentada e criticada em [4], [5] e [6] sem que tenha sido estabelecido um consenso a respeito. Segundo [7] a possibilidade dos pesquisadores de IA chegarem a um consenso a respeito desta definição é a mesma de se chegar a uma definição de "inteligência artificial". Em [6] encontramos uma comparação do problema das diversas definições de agentes com o teste *Rorschach*. Neste teste, derrama-se tinta em um pedaço de papel e pede-se a um grupo de pessoas para descrever o que vê. Alguns descreverão árvores, outros flores, outros ainda nuvens ou objetos diversos. Na verdade não há nada além de manchas de tinta. Segundo o autor, um leitor ao se deparar com uma coleção de artigos sobre agentes poderá pensar que se trata de um tipo de teste Rorschach e, em um arroubo de ceticismo, concluir que nada de concreto se relaciona ao assunto. De acordo com [8], existe um senso comum nas definições, como um representante que age em benefício de outros. Assim é estabelecida a primeira propriedade fundamental dos agentes:

- Agir em benefício de outra entidade.

Os mesmos autores definem como segunda propriedade fundamental a autonomia e, a seguir, a pró-atividade (tradução de *proactivity* - A ausência de pró-atividade seria passividade. O pró-ativo não age somente em resposta ao ambiente) e reatividade (tradução de *reactivity* - Capacidade de perceber mudanças no ambiente e reagir de modos diversos). Finalmente estabelecem que os agentes podem possuir outros atributos, citando alguns considerados chave como aprendizagem, cooperação e mobilidade. Assim, a definição pode ser sumariada como :
"Um agente é uma entidade computacional que:

- age em benefício de outras entidades de maneira autônoma;
- executa suas ações com algum grau de pró-atividade e/ou reatividade;
- exibe algum nível de outros atributos como aprendizagem, cooperação e mobilidade".

2.2.1 Aplicações para agentes

Segundo [8], precisamos de agentes porque:

- mais e mais tarefas do dia-a-dia são baseadas em computadores;
- o mundo se encontra no meio de uma revolução da informação, resultando em grande quantidade de informações dinâmicas e não estruturadas;
- gradualmente mais usuários são inexperientes;
- conseqüentemente os usuários requerem agentes para ajudá-los a entender o mundo complexo que estamos criando.

2.2.2 Agentes Móveis

Um tipo específico de agentes, denominado usualmente como agentes móveis, agentes itinerantes[9] ou agentes transportáveis[10], tem sido objeto de várias pesquisas e é sobre esta classe de agentes que trata este trabalho.

As definições de agentes móveis seguem um padrão, visto que ressaltam a possibilidade de migração de código e estado, bem como a autonomia de execução.

Em [11] um agente móvel é definido como aquele que pode se mover em uma rede heterogênea, de um nó para outro, sobre seu próprio controle, interagindo com recursos existentes ou com outros agentes, tipicamente retornando ao local de origem quando sua tarefa estiver concluída.

Pesquisadores de I.A. [8] buscam uma definição mais refinada, acreditando que a habilidade de se transportar a outros locais não é suficiente para descrever um agente móvel. É claro que se espera mais de um agente móvel do que a simples mobilidade, mas determinar o quanto e como um agente será mais ou menos inteligente é uma tarefa pretensiosa e podemos encontrar uma análise coerente deste tipo de problema em [12]. O autor, renomado pesquisador de I.A., enfoca os avanços feitos na área, mas questiona os resultados obtidos em relação ao problema de agentes inteligentes. Passaremos então a tratar dos agentes móveis sem levar em conta seu grau de inteligência e somente suas características citadas em [13], quais sejam:

- autonomia;
- comunicação;
- mobilidade;
- auto-iniciação (tradução de *self-starting*);
- orientado a objetivo.

A seguir analisaremos as vantagens e possíveis aplicações, linguagens utilizadas, requisitos de segurança e, finalmente, algumas implementações de sistemas de agentes móveis. A partir deste ponto consideraremos o uso do termo agentes somente no escopo dos agentes móveis e uma agência como o sistema de suporte aos agentes, ou seja, o ambiente que permite ao agente migrar e executar sua tarefa.

2.2.3 Aplicações para agentes móveis

Veremos nesta seção quais as vantagens advindas pelo paradigma de agentes móveis e quais os problemas que podem ser resolvidos de modo mais eficiente utilizando-se os agentes móveis.

As principais vantagens deste paradigma são:

- eficiência - tanto porque, em certos casos consomem menos recurso de rede já que migramos a computação até os dados, quanto pela possibilidade de migrar para onde existam melhores recursos (cpu, memória, etc);
- redução de tráfego na rede - visto que a maioria dos protocolos de comunicações envolvem várias interações, especialmente quando medidas de segurança estão habilitadas, com o uso dos agentes teremos estas interações acontecendo localmente;
- autonomia assíncrona - tarefas podem ser dadas a agentes, e estes operarão assíncrona e independentemente do programa que os enviou;

- controle de atividades em tempo real - como os agentes executam localmente, evita-se o problema de latência, intolerável no controle de aplicações de tempo real;
- robustez e tolerância a falhas - a habilidade de reagirem dinamicamente em situações adversas tornam os agentes mais propícios a um comportamento tolerante a falhas;
- desenvolvidos de modo a suportar ambientes heterogêneos podem atuar como um mecanismo que aumente a interoperabilidade entre estes ambientes;
- paradigma de desenvolvimento conveniente - o projeto e implementação de sistemas distribuídos pode se tornar mais fácil com o uso de agentes móveis pois estes são inerentemente distribuídos e assim possuem um abstração natural de um sistema distribuído.

Através destas vantagens, podemos perceber de antemão algumas aplicações para estes agentes e entendemos o porquê deste paradigma estar sendo tão pesquisado e discutido. Antes de passarmos aos trabalhos que demonstram aplicações para agentes móveis, vejamos algumas destas aplicações:

⇒ Comércio eletrônico;
⇒ Coleta de dados em locais remotos;
⇒ Monitoração de informações remotas;
⇒ Processamento paralelo;
⇒ Gerência de redes ou sistemas distribuídos;
⇒ Computação móvel.

Em [14] o autor aborda o uso de agentes inteligentes para gerência de sistemas e em [15] são apontados os agentes móveis como uma possível solução para a gerência de redes grandes, além de outros problemas como a recuperação de informações e o comércio eletrônico. O comércio eletrônico tem sido uma das áreas de maior pesquisa para agentes móveis e é citado também em [16]. Outra área na qual agentes móveis parecem ser solução para diversos problemas é a computação móvel e isto é citado em [9], [11] e [15].

Embora não haja um problema específico que somente agentes solucionariam, há que se reconhecer que este paradigma trará muitos benefícios e soluções melhores aos problemas das áreas citadas e de muitos outros que ainda poderão surgir.

2.2.4 Linguagens Utilizadas

Segundo [13], qualquer linguagem pode ser usada, mas é evidente que as linguagens interpretadas são mais adequadas às características dos agentes, principalmente sua natureza migratória e autônoma. Entre as mais utilizadas para a implementação de agentes temos: Tcl, que normalmente tem seu interpretador estendido para suportar a migração; Telescript, desenvolvida pela General Magic e utilizada no sistema de agentes Tabriz; e Java, linguagem desenvolvida pela Sun Microsystems que tem sido a mais adotada para agentes móveis pela sua facilidade de uso e constante desenvolvimento. Para corroborar esta afirmação basta dizer que a General Magic abandonou seu sistema baseado em Telescript em favor do desenvolvimento de outro baseado em Java. Uma das características mais importantes de Java é a possibilidade de, sem muito esforço, capturar o estado de um objeto e serializá-lo através do Object Serialization [17]. Além disto, oferece também um protocolo para o transporte deste código através do RMI (Remote Method Invocation).

2.2.5 Requisitos de Segurança

Se analisarmos inicialmente os requisitos de segurança dos agentes e das agências através dos aspectos tradicionais deste estudo veremos que o fundamento básico sobre o qual todos os aspectos de segurança são criados é a **autenticação**, ou seja, a habilidade de se verificar e assegurar a identidade de um processo ou requisição. Tanto os agentes quanto as agências requerem autenticação um do outro. Em um mundo altamente distribuído como a Internet o desafio é assegurar a possibilidade de autenticação onde esta for necessária, requerendo assim, uma apropriada infra-estrutura de autenticação.

Políticas precisam ser definidas de modo a estabelecer aos agentes quais nós podem ser visitados e às agências quais agentes serão recebidos e a que recursos (CPU, memória, arquivos, etc) terão acesso. Ou seja, políticas de autorização devem mapear identidades autenticadas a funções, requisições e recursos. Políticas de auditoria podem ajudar a agentes e agências (ou a seus proprietários) a reconstruir o passado em caso de tentativa de quebra ou quebra efetiva de segurança.

Os aspectos de confidencialidade e integridade são mais facilmente tratados nas agências através do uso de técnicas padrões dos sistemas operacionais ou por mecanismos criptográficos dos quais os agentes não possuam as chaves utilizadas. No caso dos agentes isto é problemático já que estes dependem das agências e sobre elas executam seu código. O agente pode ter a confidencialidade assegurada em algum dado específico criptografando-o com uma chave não disponível quando executando em uma agência classificada como não confiável. Em relação à integridade, se um agente resolve executar em uma agência não confiável, ou seja, corre o risco de não ter preservado seu código e dados, esta integridade deverá ser verificada e recuperada por algum sistema confiável.

Em [18] encontramos descrições de possíveis soluções para prover segurança a agentes móveis, como por exemplo:

- assinatura iterativa de dados - após visitar uma agência, o estado do agente deverá ter sido modificado. A fonte do novo estado (agência visitada) deverá assinar este dado. Isto resulta em um conjunto acumulativo de dados e assinaturas.
- estabelecimento de domínios confiáveis ;
- encapsulamento de políticas de segurança nos agentes - em uma situação de falha ou desconexão da rede, os servidores de autorização podem não estar acessíveis, requerendo então que as políticas de autorização estejam presentes no local.

Basicamente as agências devem possuir mecanismos de proteção contra agentes maliciosos estabelecendo limites de acesso através de credenciais fornecidas a cada tipo de agente e os agentes, por sua vez, devem ter protegidos tanto seu código quanto seus dados contra ataques efetuados por outros agentes ou por agências não confiáveis.

2.3 Agentes Móveis em CORBA

A especificação da Arquitetura de Facilidades Comuns [19] prevê serviços qualificados como Gerência de Tarefas e entre estes, a Facilidade de Agentes Móveis. Visando especificar este serviço, o OMG emitiu em novembro de 1995 um *Request for Proposal* [20]. Com as diferentes abordagens apresentadas, seguiu-se uma fase de re-submissões e busca de consenso de modo que a Facilidade de Agentes Móveis se transformou em Facilidade de Interoperabilidade de Sistemas de Agentes Móveis [21]. Com isto, a submissão final se tornou genérica a ponto de não tratar os agentes como Objetos CORBA e não definir mecanismos de transporte dos agentes por um ORB. De fato, outro RFP foi emitido visando a especificação da passagem de objetos por valor em CORBA - *Object by Value* RFP [22] - que até então somente permite a passagem por referência. Porém a passagem de objetos por valor seria um mecanismo a ser utilizado por um suporte a agentes móveis. Os agentes móveis são mais que objetos passados por valor pois pressupõem autonomia de execução e as demais características citadas anteriormente.

O trabalho da submissão feita conjuntamente pela IBM, Crystaliz, General Magic, Open Group e GMD Fokus demonstra uma oscilação em relação aos objetivos, visto que o documento final deixa claro que a interoperabilidade tratada se refere a sistemas desenvolvidos em uma mesma linguagem e não trata a interoperabilidade de sistemas desenvolvidos em linguagens diferentes. Porém, em [23], o motivo alegado para não prover mecanismos de codificação para o transporte de agentes foi exatamente a independência de linguagens de programação. O presente trabalho se diferencia do enfoque do OMG e demonstra que, mesmo em um ORB que não permita ainda a passagem de objetos por valor, é possível se ter agentes que, mesmo sendo objetos CORBA, migram utilizando o protocolo

IIOP do ORB. Assim estes agentes, com suas referências de objetos, usufruirão de todos os benefícios oferecidos pelo *broker* como a transparência de localização e acesso.

3 Modelagem e Projeto

Na fase inicial do trabalho de modelagem, estabelecemos os seguintes objetivos básicos:

- Os agentes serão objetos CORBA, ou seja, terão sua interface definida em IDL de modo a serem acessíveis por qualquer objeto cliente e acessarem qualquer outro objeto CORBA, independente de linguagem de programação;
- Os agentes serão identificados por sua referência de objeto e assim também localizados transparentemente pelo *broker*;
- O suporte tratará qualquer número de agentes, de qualquer tipo, em *threads* independentes e com igual prioridade;
- O suporte permitirá a captura do estado do agente, representado pelas variáveis globais;
- Para a mobilidade será utilizado o protocolo IIOP;
- O suporte permitirá a auto-iniciação e a autonomia de execução dos agentes, bem como a possibilidade de se terminar agentes através de comandos externos aos mesmos.
- O suporte permitirá a modificação do itinerário de um agente lançado;
- O suporte será capaz de executar simultaneamente em um dado host, agentes recebidos de diferentes hosts e agentes criados neste host.

Podemos notar que, destes objetivos iniciais, advêm várias outras características para a modelagem. Entre elas a necessidade de se definir uma interface em IDL para os agentes e a de permitir a atualização da referência de objeto quando o agente se mover. Esta atualização da referência de objetos funcionará como um relocador que permitirá que a localização e a comunicação com os agentes ocorra de modo transparente, independente desta localização. Em certos casos, quando for importante o conhecimento preciso da localização dos agentes, o registro da referência atual permitirá esta localização. Para sistemas sob o NFS, em relação ao registro dos agentes, tal somente é necessário no domínio do ORB visto que o *broker* utiliza o NFS para instanciar os objetos em qualquer *host* de seu domínio. Para o caso de o agente ser enviado para um *host* fora deste domínio, no qual não estiver registrado, o suporte proverá um método para que este registro ocorra, transparentemente ao cliente.

Neste trabalho não estabelecemos como meta o tratamento de problemas relacionados a segurança que, embora vital a este paradigma, será abordada em trabalhos futuros. O enfoque primordial é a migração de objetos CORBA, porém o modelo não inviabiliza os mecanismos de segurança estudados atualmente. A Figura 3 apresenta um possível cenário (as setas indicam o sentido da migração).

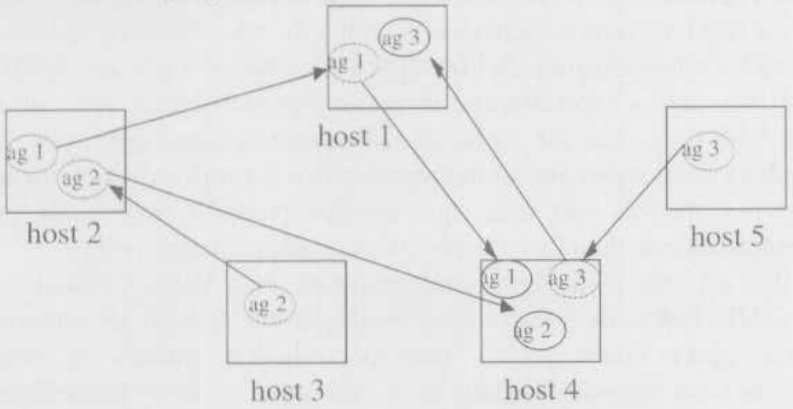

Figura 3 - Cenário de uma possível migração de agentes.

Neste cenário o agente ag1 foi criado no host 2, migrou para o host 1 e posteriormente para o host 4, onde se encontra ativo. As linhas tracejadas apenas representam a passagem de um agente por um host e nesta representação o agente não mais existe nestes hosts. Podemos observar que um host deve ser capaz de receber diversos agentes ao mesmo tempo em que poderá estar criando agentes. Por exemplo no host 2 o ag1 poderia estar sendo instanciado simultaneamente com a chegada de ag2.

A interface IDL do suporte (**MAF.idl**), transcrita no Anexo A, define também uma interface para os agentes e esta deve ser então implementada em uma classe que será especializada para cada tipo de agente.

Pelo cenário exposto acima, verificamos que o suporte deve ser mais simples quando apenas recebe agentes do que quando instancia um agente atendendo a uma aplicação. Desta forma, duas classes distintas foram definidas:

MAF1 - permite a criação de agentes e controla seu histórico ;

MAF2 - utilizada para receber agentes já criados em outro *host*.

O suporte deve prover meios de comunicação não somente entre objetos CORBA mas também entre os agentes e aplicações cliente que porventura não sejam objetos CORBA, ou seja, não estejam registradas no ORB.

Inicialmente é necessária a definição dos participantes do sistema. Em princípio, definimos três: o suporte propriamente dito, o usuário do suporte que seria a APLICAÇÃO e o desenvolvedor de agentes.

Assim, chegamos ao modelo estático apresentado na Figura 4.

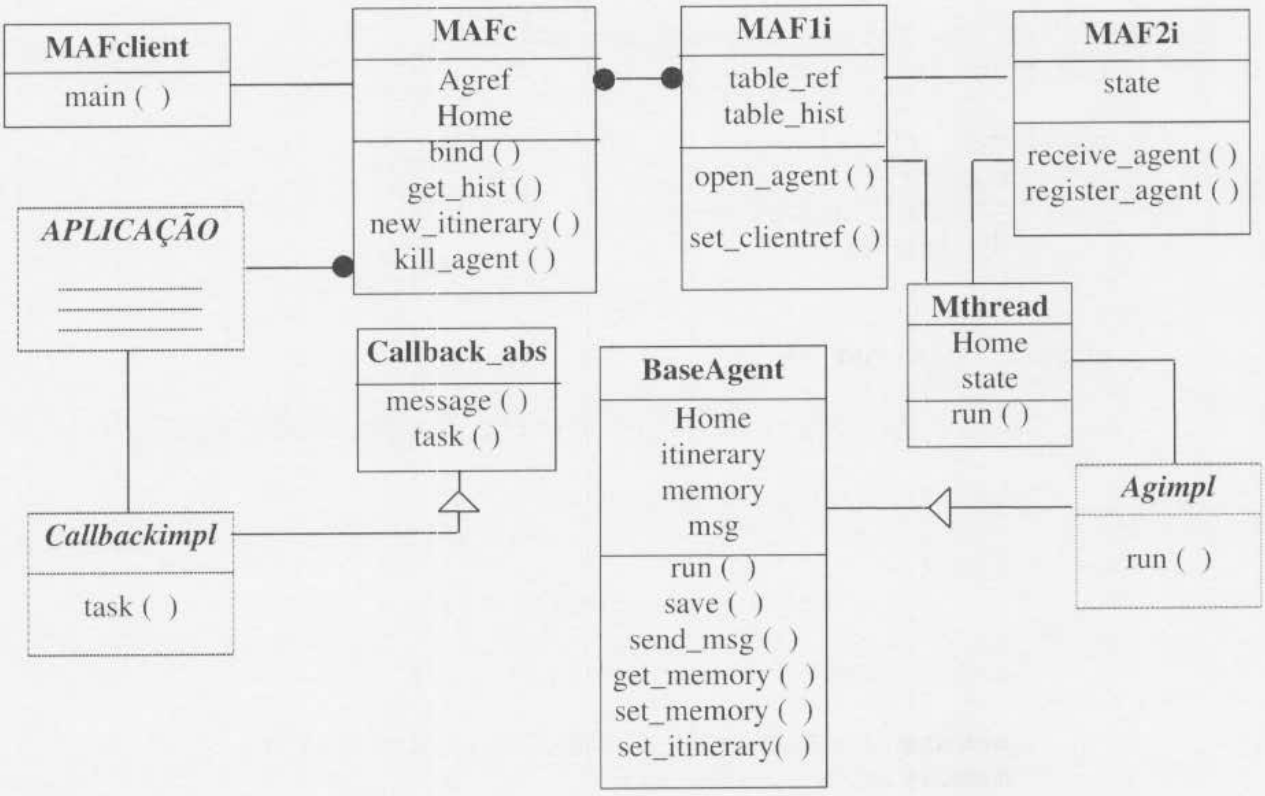

Figura 4- Modelagem do Suporte a Agentes Móveis.

Como o suporte irá trabalhar com agentes que possuem uma interface IDL definida, visando facilitar o desenvolvimento destes agentes existirá uma classe denominada **BaseAgent** que implementará a interface IDL e será especializada pelo desenvolvedor de um agente.

A classe específica do agente é mostrada na modelagem em pontilhado e chamada **Agimpl**, mas tanto o nome quanto a implementação ficam a critério do desenvolvedor. **BaseAgent** é uma classe abstrata e, ao ser especializada deve ter o método *run*

implementado. Este método trará o código a ser executado para cada tipo específico de agente. Em **BaseAgent** existem as seguintes variáveis de instância:

- `Home`: trata-se de referência de objeto do **MAF1i** que controla este agente. É usada para que a cada *host* o agente armazene um histórico que possa vir a ser recuperado posteriormente.
- `itinerary`: usada para armazenar o itinerário do agente;
- `memory`: variável genérica o suficiente para armazenar qualquer tipo e quantidade de dados usados ou coletados pelo agente e;
- `msg`: também genérica de modo a permitir que qualquer tipo de dado possa ser passado como conteúdo de uma mensagem.

Em **BaseAgent** já existem métodos para atuar nas variáveis `itinerary` (*set_itinerary*) e `memory` (*set_memory e get_memory*), além do método *save* utilizado para capturar o estado do agente quando este for se mover e o *send_msg* para o envio de mensagens ao cliente.

A implementação de um agente é bastante simples como mostra o exemplo abaixo:

```
package manager;

import MAF.*;

public class Collector extends BaseAgent {
     int count;

     collector (){
          super();
          _ObjRef MngObj = null;
          String s;
     }

     public void run () {
          try {
               MngObj = Obj._bind(":",_CORBA.Orbix.myHost());
          }
          catch (SystemException e){ ...}

          try {
               s = MngObj.get_results();
          }
          catch (SystemException f) { ...}

          memory.addElement(_CORBA.Orbix.myHost());
          memory.addElement(s);
     }
}
```

Este agente atua coletando informações em diferentes hosts, contatando localmente um objeto (outro objeto CORBA- MngObj). O host visitado e a informação coletada serão colocadas em `memory` para posterior utilização por uma aplicação de gerência de sistemas ou pelo usuário que lançou o agente.

Após o agente ter sido implementado e registrado no ORB, o suporte estará pronto a controlar o envio deste agente por parte de uma APLICAÇÃO. Para tal, existirá uma classe a ser instanciada para cada agente lançado, denominada **MAFc**. A instância de **MAFc**

promoverá, transparentemente ao usuário, através do método *bind*, o controle da criação do agente no primeiro *host* de seu itinerário pela chamada de *open_agent* em um objeto **MAF1i**.

O ciclo de vida de um agente inicia e é controlado pela classe **MAF1i**. Existirá somente uma instância desta classe por *host* e esta poderá criar quantos agentes forem solicitados. Esta classe se reveste de especial importância pois será o elo de ligação permanente dos agentes "nascidos" naquele *host* com a APLICAÇÃO. O histórico de informações dos agentes, bem como o controle das referências de objeto será efetuado pela instância desta classe e para isto existem, respectivamente, as tabelas `table_hist` e `table_ref`. O método *set_clientref* em **MAF1i** será usado para estabelecer a referência do cliente caso venha a ser necessário o envio de mensagens para este.

Em **MAFc** ainda existirão métodos para a obtenção do histórico do agente (*get_hist*), para modificar o itinerário (*new_itinerary*) e para finalizar agentes quando necessário (*kill_agent*). Naturalmente cada **MAFc** terá a referência do agente criado (`Agref`) e a do **MAF1i** que o controla (`Home`).

Nos *host*s subseqüentes, o agente será recebido e instanciado por um objeto da classe **MAF2i**. Também somente será necessária uma instância desta classe por *host* e esta tratará o recebimento de vários agentes. A relação 1 para muitos, demonstrada na Figura 6, se refere ao caso de agentes criados em um *host* possuírem diferentes itinerários e assim, um mesmo **MAF1i** se ligaria a diferentes **MAF2i**. A principal diferença entre as classes MAF se encontra no fato de a classe **MAF2i** ser bem mais simples já que nada armazena sobre os agentes. Esta classe possui apenas a variável `state` que é o estado de execução do agente e os métodos para receber o agente e instanciá-lo (*receive_agent*) e para, caso necessário, registrar o agente no ORB (*register_agent*).

A execução do código do agente será controlada por um *thread* instanciado por um objeto **MAF1i** ou **MAF2i**, denominado **Mthread**. Esta execução em *threads* independentes, além de permitir a instanciação simultânea de vários agentes em um mesmo *host*, propicia também o escalonamento equânime destes agentes. Como cada agente estará associado a um **Mthread**, esta classe deverá possuir as variáveis `Home` e `state` de seu agente, além do método *run*, típico de *threads*.

Caso seja necessário o recebimento de informações diretamente dos agentes, haverá a necessidade da implementação pelo usuário, de uma classe aqui denominada **Callbackimpl**, que será uma especialização de **Callbackabs**. Esta última estará disponível e tratará o recebimento das mensagens de agentes pelo método *message* e possuirá o método abstrato *task* a ser implementado na especialização desta classe. Assim como a APLICAÇÃO, a classe **Callbackimpl** ficará a critério do usuário e por isto aparece na modelagem em linhas tracejadas. Os detalhes da comunicação via ORB entre os agentes e a APLICAÇÃO ficam transparentes ao usuário pela classe **Callbackabs**. Embora o tipo da mensagem venha a ser padronizado (a variável `msg` em **BaseAgent**), o conteúdo dependerá do agente sendo utilizado e as ações pertinentes dependerão de cada APLICAÇÃO.

Por fim, estabelecemos uma classe **MAFclient** que permitirá o lançamento de agentes via comando de teclado como por exemplo:

```
MAFclient <agente> <host1> <host2> <host3> ...
```

Esta classe será útil para aqueles agentes que não necessariamente retornam informações e não precisam estar ligados a uma aplicação. Ela instanciará um **MAFc** apenas para executar o método *bind*.

Independentemente da APLICAÇÃO, o agente irá morrer quando terminar seu itinerário. O **MAF1i** que o criou ficará ativo enquanto for necessário. O método *kill_agent* somente será usado quando se quiser matar o agente antes do término de seu itinerário.

4 Implementação

A plataforma Multiware está sendo implementada em estações IBM RS/6000 e Sun Sparc, com os sistemas operacionais AIX e Solaris, conectados por redes ethernet, fast ethernet e FDDI. Este ambiente está disperso em dois laboratórios, inter-conectados por um enlace de 10 Mbits. Portanto, o suporte deve ser portável para tais sistemas heterogêneos.

Adotamos Java como linguagem dos agentes pela portabilidade de seu código e por possuir características que permitem a captura e recuperação do estado dos objetos sem a necessidade de alterar o interpretador da linguagem. Além disto, dentre as linguagens comumente utilizadas para agentes móveis, é a única que já possui o mapeamento para IDL especificado e ORBs comerciais que já trazem compiladores IDL-Java.

Os ORBs atualmente instalados nos laboratórios são o Orbix 2.1 MT e OrbixWeb2.0.1, ambos da Iona Technologies Ltd.. Como o suporte será para agentes escritos em Java, este foi implementado e registrado na OrbixWeb. Isto não impede que os agentes sejam lançados por ou se comuniquem com objetos implementados em outras linguagens como os Objetos CORBA da Orbix 2.1 (mapeamento para C++) ou de outro ORB futuramente instalado, visto que será usado o protocolo IIOP. No exemplo de implementação citado o objeto a ser consultado (MngObj) poderá ter sido implementado em C++.

A implementação da classe **BaseAgent** já prevê a serialização do estado de execução dos agentes através do método ***save*** que, usando o *Object Serialization* de Java, retorna um *array* de *bytes* que representa este estado. Na OrbixWeb2.0.1, um *array* de *bytes* precisa ter seu tamanho definido na interface IDL. Como a interface ***Agent*** deve ser genérica optamos por utilizar o gabarito denominado *sequence* em [1, p.3-28] e implementado neste ORB possibilitando a definição de um tipo sem tamanho definido. Assim denominamos um tipo ***Stringseq*** a ser usado para a passagem de *arrays* de *strings* como no caso do itinerário, por exemplo, e o ***Byteseq*** para a passagem de *arrays* de *bytes* como o estado de objetos e o conteúdo de mensagens de agentes, por exemplo. Esta foi uma maneira simples de integrar a capacidade de serialização de Java e passar este estado como um parâmetro pelo ORB. Da mesma maneira serão passadas as classes, pois ao registrar o agente no ORB, o desenvolvedor fornece o caminho (*PATH*) do *package* onde se encontram as classes e estes arquivos também são facilmente serializados em Java e passados pelo ORB da mesma maneira que foi feito com o agente. A passagem das classes somente será necessária se o agente estiver sendo enviado para fora do domínio do ORB onde foi registrado e se, neste outro ORB, este agente não tiver sido registrado. Por isto e visando aumentar a eficiência do suporte, em princípio somente é enviado o estado através da chamada do método ***receive_agent***. Caso o ORB não possua registro deste agente, a exceção retornada será tratada de forma a enviar as classes também através da chamada de ***register_agent*** que, no outro ORB, através de métodos existentes no *daemon* da OrbixWeb [24] (este *daemon* é também um Objeto CORBA), irá prover o registro deste agente.

Tanto para a passagem de classes quanto para a passagem do estado são utilizados arquivos temporariamente criados no destino. Estes arquivos são automaticamente apagados após a recuperação e instanciação dos agentes.

O desenvolvimento de Objetos CORBA pressupõe a criação de uma classe denominada servidor do objeto. Este servidor não é a implementação de objetos que provê o serviço oferecido e sim apenas uma classe responsável por instanciar o objeto e atuar como parte da abstração do esqueleto estático do ORB. Como o mesmo agente (que possui o mesmo servidor) será uma vez criado e outras vezes recuperado, houve a necessidade de se definir como fazer a correta seleção na implementação do servidor. Neste caso, não é possível a criação de uma classe base para ser especializada e optamos por definir um modelo a ser usado na construção dos servidores. Trata-se de parte do manual de utilização do sistema.

Na definição das referências de objetos na OrbixWeb, é levado em conta o *host* no qual o objeto é instanciado. Assim, quando o agente se movesse, perderia sua identidade. Tivemos então que criar um relocador de objetos e por isto a necessidade do mapeamento de

referências em **MAF1i**. Além disto, como o agente muda constantemente de referência e esta tabela será freqüentemente atualizada, precisamos criar um mecanismo que evitasse a chamada a um agente que já tivesse em outro *host*, com outra referência portanto. Desta forma, transparentemente ao cliente, qualquer chamada ao agente criado consultará antes a `table_ref` e será encaminhada ao agente corretamente.

Para evitar que uma chamada seja encaminhada ao mesmo tempo em que o agente se move, o que provocaria um erro, criamos um mecanismo de exclusão mútua de forma a impedir que o agente se mova caso a `table_ref` seja consultada e impedir a consulta desta quando o método *save* for executado.

Na Figura 5, demonstramos o lançamento de dois agentes que, partindo de um mesmo *host*, perfazem itinerários diferentes.

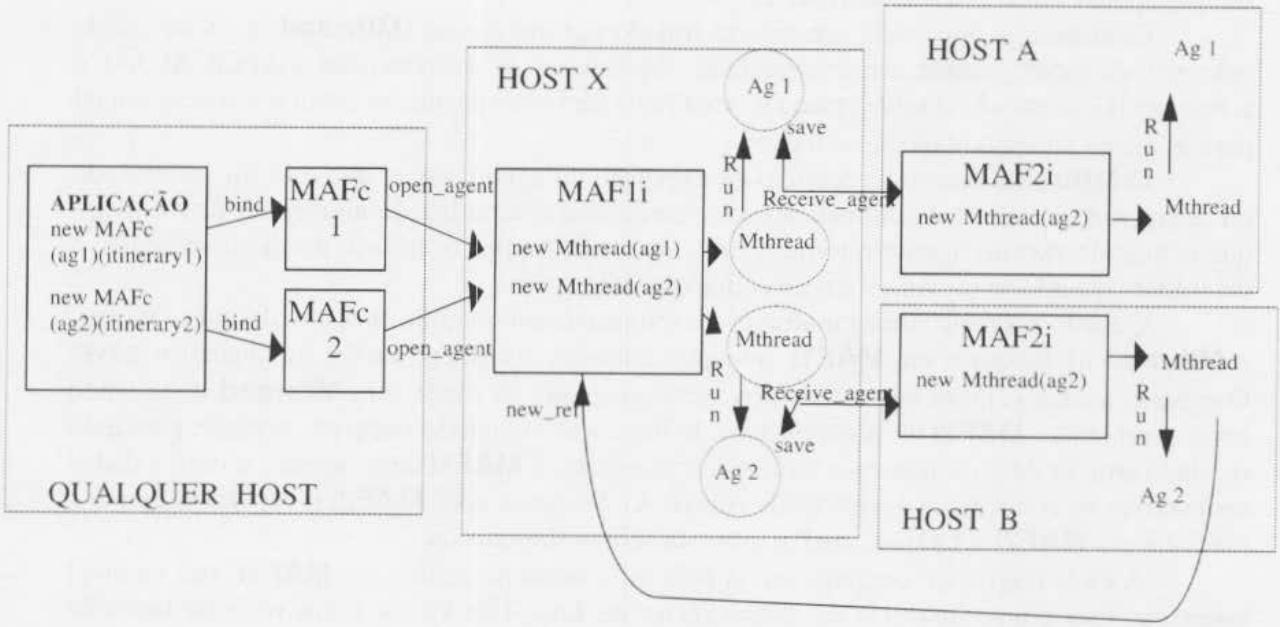

Figura 5 - Cenário do lançamento de dois agentes.

Uma APLICAÇÃO lança dois agentes (1 e 2) sendo que o agente 1 possui o itinerário composto pelos *hosts* X e A e o agente 2 possui o itinerário composto por X e B. Em X, os agentes são criados em *threads* independentes e executam de maneira autônoma.

Na Figura 5, o Ag1 já está ativo no host A e o Ag2 no host B. O **Mthread** criado no host B para a execução de Ag2 está atualizando a referência de objeto deste agente em seu **MAF1i** (no host inicial X).

Cabe ressaltar que os nomes da máquinas são utilizados para definir o itinerário, ao contrário de URLs como previsto na submissão feita ao OMG. O uso de nomes de máquinas é mais estável que as URLs pois as últimas são freqüentemente modificadas. A única preocupação do usuário é conhecer o domínio no qual está instalado o ORB. Caso o agente tenha que sair do ORB deverá ser utilizado o nome completo da máquina.

Por exemplo, nos nossos testes utilizamos o laboratório da Faculdade de Engenharia Elétrica e de Computação e o do Instituto de Computação. Os ORBs foram instalados nos seguintes domínios:

IC : `dcc.unicamp.br`

FEEC : `dca.unicamp.br`

Assim, para um agente ser lançado do IC, percorrer o itinerário xingu (IC), pinheiros (IC), caolho (FEEC), aracati(FEEC) e regressar a xingu (IC), será necessário passar o seguinte *array* de *strings* que comporá o itinerário:

xingu - pinheiros - caolho.dca.unicamp.br - aracati - xingu.dcc.unicamp.br

Este itinerário pode ser modificado ou substituído em qualquer *host*, mantendo o cuidado de assegurar que o ORB irá conhecer as máquinas. Caso um usuário queira evitar problemas, o fornecimento sempre do nome completo da máquina previne qualquer erro.

Caso um *host* não seja localizado ou por ter sido fornecido o nome errado ou por falha na conexão ou na própria máquina, o agente irá parar e terminar como se o *host* atual fosse o último de seu itinerário. Através do uso do histórico em **MAF1i** é possível a verificação do caminho percorrido.

Conforme mostrado na Figura 5, para cada agente existirá um **MAFc** que em seu método *bind* se ligará ao **MAF1i** do *host* inicial do itinerário. Se já houver um objeto **MAF1i** instanciado, a criação do novo agente será também tratada por este **MAF1i**. Assim teremos sempre apenas uma instância de **MAF1i** por *host*.

Conforme já dito, cada agente será tratado por um *thread* (**Mthread**) e ao ser criado pelo método *open_agent*, sua referência de objeto além de ser retornada à APLICAÇÃO, é armazenada em uma *hashtable* (`table_ref`) que fará o mapeamento desta referência inicial para as atuais, quando o agente se mover.

O **Mthread** determina o início de execução do agente assim que este for instanciado ou recuperado e, também automaticamente, determina a serialização através do *save*, assim que o método *run* do agente retorne. Como este método fica a critério do desenvolvedor, a decisão de quando se mover estará no código de cada agente.

Quando o agente sinalizar através do retorno de seu método *run*, que deseja se mover, o **Mthread** irá bloquear em **MAF1i**, qualquer consulta a `table_ref` e irá chamar o *save*. O próprio agente captura seu estado e retorna um *array* de *bytes* para **Mthread**. Este então invoca, no objeto **MAF2i** do próximo *host* do itinerário, o método *receive_agente*, passando o estado através de uma *sequence* de *bytes*, o itinerário, o **MAF1i** deste agente, e outros dados necessários ao controle do agente (vide Anexo A). Somente após **MAF2i** recuperar o agente e atualizar em **MAF2i** a `table_ref` é que esta será desbloqueada.

A cada migração, o agente envia para uma outra *hashtable* em **MAF1i**, sua variável `memory`. Esta é uma instância da classe *Vector* de Java. Um *Vector* é um vetor de tamanho indefinido de objetos.

Assim, qualquer tipo e quantidade de objetos pode ser ali armazenado ficando a critério dos desenvolvedores dos agentes o que armazenar e em quais posições pois a classe Java também permite inserir e retirar elementos de qualquer posição.

Cabe ressaltar que em Java, os tipos básicos também possuem representação através de objetos como, por exemplo, um inteiro (`int`) pode ser representado como uma instância da classe ***Integer***.

Esta classe *Vector* de Java permitiu não somente a generalização das variáveis dos agentes, mas também do conteúdo das mensagens.

A contribuição do nosso suporte foi prover métodos para que um Vector fosse transparentemente transformado em uma *sequence* de *bytes* e vice-versa.

O desenvolvedor de um agente não se preocupa com as características relacionadas a IDL e tão somente trabalha em Java.

Até este ponto estamos trabalhando apenas com a hipótese de que o cliente (APLICAÇÃO) irá consultar o **MAF1i** para obter informações dos agentes.

Porém, em nossa análise, previmos três tipos de situações mostradas na Figura 6.

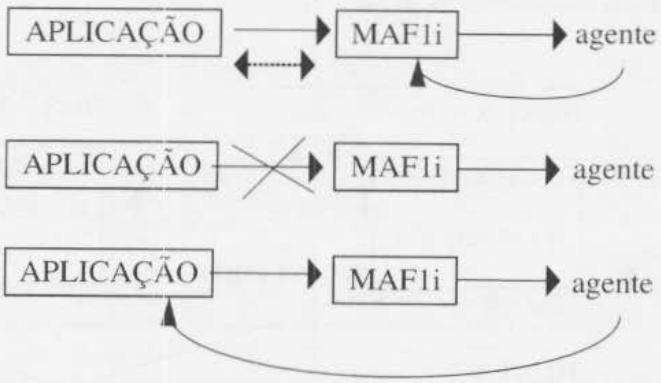

Figura 6 - Situações de uso de agentes

O primeiro caso é o que foi tratado até aqui, ou seja, após lançado o agente, o cliente pode inclusive se desconectar da rede como acontece na computação móvel e depois, ao se conectar, pode buscar informações no **MAF1i**. O segundo caso é quando o agente a ser lançado não precisa retornar nenhuma informação e, por ser mais simples, está também atendido. O terceiro caso prevê a necessidade do cliente, para a continuidade de seu processamento, receber informações dos agentes. Para este último caso previmos na modelagem um modo de clientes, mesmo não sendo Objetos CORBA, receberem mensagens dos agentes.

A implementação desta funcionalidade se tornou simples pelo fato de já exitir na OrbixWeb o mecanismo de *Callback* [25]. Criamos então a classe abstrata **Callbackabs** que para cada tipo de aplicação cliente, é especializada. Em **Callbackabs** existe o método ***message*** que transforma para o cliente a variável `msg` recebida de *sequence* de *bytes* para a classe *Vector* de Java, assim como em **BaseAgent** o método ***send_msg*** transforma o *Vector* em *sequence* de *bytes*. Assim, nem o desenvolvedor de agentes e nem o usuário precisam ser profundos conhecedores do ORB. A especialização de **Callbackabs** precisará implementar apenas o método ***task*** que possui como parâmetro o *Vector* recebido. Este método, conforme o próprio nome indica, estabelecerá a tarefa a ser feita com o conteúdo da mensagem.

Na OrbixWeb, um Objeto CORBA somente atende a uma invocação por vez. Assim sempre que for necessário o atendimento de mais de uma chamada, tanto o desenvolvedor de agentes quanto o usuário deverão instanciar um *thread* denominado **EventProcessor**. Este *thread* tratará o recebimento de mensagens pelo *Callback*, para o cliente. Para os agentes, ele será útil quando quiser a invocação de métodos nos agentes durante sua execução, como por exemplo um ***set_itinerary*** ou ***get_memory***.

Por fim, vimos que embora atendendo a todos os tipos de aplicações vislumbrados, ficava a necessidade de se determinar quando e como o objeto **MAF1i** iria morrer. Este deve ficar ativo enquanto os agentes criados por ele estiverem vivos e, ainda mais, enquanto o cliente assim o determinar pois um agente pode terminar e morrer sem que o cliente tenha recebido nenhuma informação. Para não deixar o **MAF1i** ativo indeterminadamente adicionamos um parâmetro em **MAFc** que, se tratando de uma variável *boolean*, faz com que o usuário determine se o **MAF1i** pode morrer (*true*) ou não (*false*). Em **MAF1i** foi criada uma nova *hashtable* de modo que para cada agente este parâmetro fosse armazenado. A condição para o **MAF1i** morrer será a de não mais existirem agentes criados por ele e para todo agente criado foi passado um *true* como parâmetro. Se algum cliente passar um *false* na criação do agente, fica compelido através de uma classe especialmente criada para tal, a passar o true quando nada mais necessitar de **MAF1i**. Esta classe citada recebeu o nome de **KillMAF**.

Para que ocorra a comunicação entre agentes ou entre agentes e outros Objetos CORBA, o agente deve levar consigo uma referência de objeto relacionada ao outro Objeto CORBA e então a invocação via ORB ocorrerá normalmente.

A única mudança para dois agentes se comunicarem é que cada agente, além da referência inicial, deverá armazenar também a referência do **MAF1i** do outro agente.

A comunicação se dará como na Figura 7.

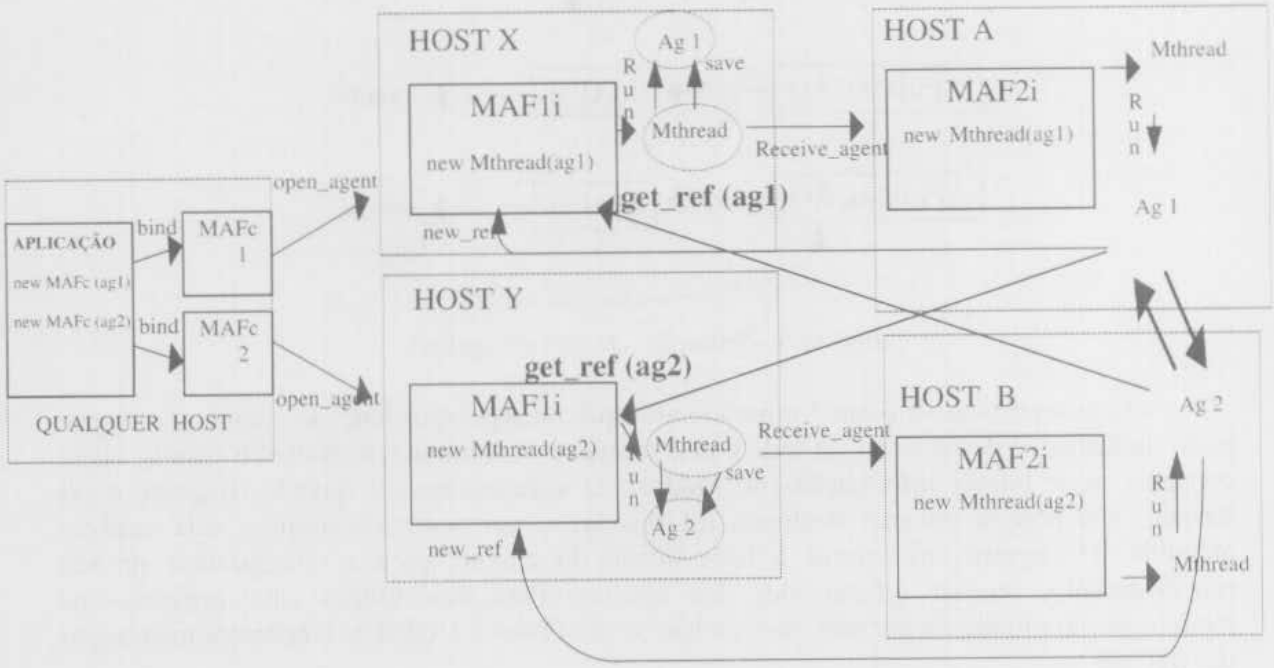

Figura 7 - Comunicação entre agentes.

O processo de atualização da referência de um agente ocorre quando este migra e é mutuamente exclusivo ao fornecimento desta nova referência a um objeto que queira se comunicar com este agente. Além disto, o envio de dados a um agente é feito somente através do *set_memory* e a coleta através do *get_memory*, que podem ser executados simultaneamente com o método *run* desde que, para isto, o agente possua um *thread* **EventProcessor** instanciado. Os métodos *get_memory* e *set_memory* estão implementados em **BaseAgent** e atuam de maneira sincronizada em `memory`.

5 Trabalhos Relacionados

Existem muitos sistemas comerciais e projetos acadêmicos de suporte a agentes móveis. Porém, embora muitos citem CORBA em determinados aspectos nenhum, com exceção ao sistema MAGNA em desenvolvimento pelo GMD-Fokus [26], trata os agentes como Objetos CORBA. Citaremos então os relacionados ao grupo que conjuntamente submeteu a proposta de especificação da Facilidade de Agentes Móveis do OMG. São eles: Aglets da IBM [27], MuBot da Crystaliz [28] , o Odissey da General Magic [29] e o MAGNA do GMD-Fokus.

Destes, o único que explicitamente cita o uso de um ORB como base é o MAGNA, além de colocar os agentes como Objetos CORBA. Porém, ainda não existe uma implementação disponível para testes e muito menos detalhes do projeto foram tornados públicos. Sabemos que também adotam Java e os agentes migram através do ORB. Os dois primeiros, Aglets e MuBot, sequer se referem a plataforma do OMG. Tivemos a oportunidade de testar o sistema Aglets e este somente permitia aos agentes acessarem outros Objetos CORBA se seu sistema de segurança estivesse desabilitado. Além disto, não possibilita alterações dinâmicas no itinerário dos agentes e utiliza, para o transporte, um protocolo baseado no HTTP e em URLs. Dos sistemas citados, o que nos parece mais flexível é o Odissey da General Magic. Embora também não utilize o ORB para o transporte, permite o uso do IIOP para a comunicação dos agentes com outros objetos. Além disto, aceita também mudanças no itinerário dos agentes.

A principal diferença do nosso MAF para todos os outros sistemas está na simplicidade e na integração do suporte com o ORB, tratando os agentes como Objetos

CORBA que, além de usufruírem das demais funcionalidades oferecidas pela plataforma do OMG, adicionam a possibilidade de migração da computação. Demonstramos assim, que é possível usar um ORB para o transporte de agentes e, de acordo com [30], o *broker* não perde em eficiência para outras formas de conexão. Para aplicações que utilizam um ORB, como as citadas em [31] e [32], nosso sistema de suporte a agentes permite um desenvolvimento e utilização mais simples e eficiente, na medida que o desenvolvedor, já conhecedor do funcionamento do *broker*, não terá o trabalho de aprendizado e adaptação de outra tecnologia ao ORB.

Outra característica do nosso suporte é que o itinerário será composto por nomes de máquinas e não por URLs, consideradas frágeis por serem freqüentemente modificadas. Este itinerário poderá ser alterado a qualquer momento, tanto pelo próprio agente quanto externamente.

6 Conclusão

Este artigo apresenta um suporte a agentes móveis para CORBA usando mecanismos do ORB para prover o meio de transporte para a migração dos agentes. Assim conseguimos aliar as vantagens oferecidas por este novo paradigma aos benefícios advindos pelo fato dos agentes serem Objetos CORBA, como a possibilidade de comunicação com outros Objetos CORBA, independentemente de sua localização e linguagem de programação, além de usufruir da interoperabilidade entre meios heterogêneos.

O uso da OrbixWeb2.0.1 propiciou-nos a facilidade do mecanismo de *callback* e Java a disponibilidade de várias estruturas prontas e que atendiam necessidades específicas como as classes *Hashtable*, *Vector*, *Threads* e a interface *Serializable*.

Os agentes móveis não se constituem em um paradigma relacionado apenas a aplicações para Internet. Em sistemas distribuídos de modo geral, a possibilidade de migrar a computação para outro ponto da rede pode propiciar o desenvolvimento de soluções mais eficientes para certos problemas como gerência de sistemas, automação de *workflow* e outros.

O atual enfoque do trabalho de especificação da Facilidade de Agentes Móveis do OMG, buscando satisfazer as diferentes tecnologias já existentes e torná-las interoperáveis fugiu do escopo inicial onde se buscava definir padrões para se ter agentes em CORBA. Ao elevar o nível da abstração, a proposta deixa em aberto o meio de se transportar os agentes, permitindo assim o uso de quaisquer mecanismos independentes de CORBA o que, na prática poderá não ser eficiente. A citada interoperabilidade entre sistemas de agentes já está comprometida pela dependência de linguagem de programação.

Concluindo, podemos afirmar que nosso suporte é flexível na medida que pode ser implementado em outro ORB que possua o mapeamento Java-IDL. Esta exigência não chega a ser comprometedora visto que, entre as linguagens mais adequadas para a implementação de agentes, a única que possui mapeamento para IDL é Java. O suporte é simples pois permite a usuários a invocação de agentes no ORB do mesmo modo que outros objetos são invocados, além de prover mecanismos para o uso dos agentes em diferentes tipos de aplicações, com ou sem o recebimento de mensagens destes, considerando ou não a desconexão do cliente e possibilitando modificações dinâmicas no itinerário dos agentes e, em relação aos desenvolvedores, é mais simples a implementação de um agente neste sistema que outro Objeto CORBA.

Agradecimentos:

Agradecemos à Marinha do Brasil, à FAPESP e ao CNPq pelo apoio financeiro na realização deste trabalho.

7 Referências Bibliográficas

[1] OMG doc formal/98-02-33 -***CORBA / IIOP 2.2 Specification*** - Full version - 1998

[2] Loyolla, W., Madeira, E., Mendes, M., Cardozo, E. e Magalhães, M. - ***Multiware***

Platform: an Open Distributed Enviroment for Multimedia Cooperative Applications - IEEE COMSAC - Taiwan - novembro 1994

[3] Franklin, S. e Graesser, A.-***"Is It an Agent, or Just a Program ?: A Taxonomy for Autonomous Agents"*** -Lecture Notes in Artificial Intelligence 1193 - pp 21-35 - 1996

[4] Castelfranchi, C. - ***To Be or Not to Be an "Agent"*** - Lecture Notes in Artificial Intelligence 1193 - agosto 1996 - pp 39-39

[5] Petrie, C. - ***"What Is an Agent?"*** - Lecture Notes in Artificial Intelligence 1193 - agosto 1996 - pp41-43

[6] Wooldridge, M. - ***"Agents as a Rorschach Test: A Response to Franklin and Graesser"***- Lecture Notes in Artificial Intelligence 1193 - agosto 1996 - pp 47-48

[7] Nwana, H. e Wooldridge, M.- ***"Software Agents Technologies"***- Lecture Notes in Artificial Intelligence 1198 - pp 59-78

[8] Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F. e Evans, R. - ***"Software Agents : A review"*** - Trinity College Dublin - maio 1997

[9] Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C. e Tsudik, G. - ***Itinerants Agents for Mobile Computing*** - IBM Research Report RC 20010 - março 95

[10] Kotay K. e Kotz, D. - ***Transportable Agents*** - Department of Computer Science - Dartmouth College - novembro 1994

[11] Gray, R., Kotz, D., Nog, S., Rus, D. e Cybenko, G. - ***Mobile Agents for mobile computing*** - Technical Report PCS-TR96-285 - Department of Computer Science - Dartmouth College - 1996

[12] Petrie, C. - ***"What's an Agent... and what's so intelligent about it ?"*** - IEEE Internet Computing - julho/agosto 1997

[13] Berbers, Y., De Decker, B. e Joosen, W. - ***Infrastructure for Mobile Agents*** - KULeuven - Department of Computer Science - 1996

[14] Muller, N. - ***"Systems Management : The Emerging Role of Intelligent Agents"*** - Strategic Information Resources - disponível em http://www.ddx.com/agents.shtml

[15] Lingnau, A. e Drobnik, O. - ***"An Infrastructure for Mobile Agents: Requirements and Architecture"*** - Fachbereich Informatik (Telematik) - Johann Wolfgang Goethe-Universität - Frankfurt 1995

[16] Merz, M. e Lamersdorf, W. - ***Agents, Services and Electronic Markets: How do they integrate ?*** - Department of Computer Science - University of Hamburg - 1996

[17] http://www.javasoft.com/products/jdk/rmi/serial/index.html

[18] Milojicic, D. - ***Alternatives to Mobile Code*** - The Case for Mobile Agents - The Open Group Research Institute - fevereiro 1997

[19] OMG doc formal/97-06-15 -***Common Facilities Architecture Specification*** - Rev. 4.0

[20] OMG doc cf/95-11-03- ***Common Facilities RFP 3- Mobile Agents Facility***

[21] OMG doc cf/97-10-05 - GMD Fokus, IBM, Crystaliz, General Magic e The Open Group ***Joint Submission - Mobile Agent System Interoperability Facilities Specification***

[22] OMG doc orbos/96-06-14 - ***Objects-by-value RFP***

[23] OMG doc orbos/97-09-02 - ***Evaluation Report on the Mobile Agent Facility Joint Submission***

[24] Iona Technologies Ltd. - OrbixWeb Programming Guide - novembro 1996

[25] Iona Technologies Ltd. - OrbixWeb Reference Guide - novembro 1996

[26] Krause,S., Silva,F., Magedanz,T., Popescu-Zeletin,R., Falsarella,O. e Méndez,C. - ***MAGNA - A DPE-based Platform for Mobile Agents in Electronic Service Markets*** - Proceedings of Third International Symposium on Autonomous Decentralized Systems - ISADS 97 - abril 1997 - pp 93-102

[27] http://www.ibm.co.jp/trl/projects/aglets/index.html

[28] http://www.crystaliz.com/logicware/mubot.html

[29] http://www.genmagic.com/html/agent-overview.html

[30] Orfali,R. e Harkey,D. - ***CLIENT/SERVER Programming with Java and CORBA*** - John Wiley -1997

[31] Schulze,B. e Madeira,E. - ***Contracting and Moving Agents in Distributed Applications***

Based on a Service-Oriented Architecture - Lecture Notes in Computer Science 1219 - 1997 - pp 74-85

[32] Ropelatto,P., Madeira,E., Schulze,B. e Vasconcellos,F. - ***Distributed Objects Management in CORBA Environment using Mobile Agents*** - IASTED International Conference on Networks and Communication Systems - NCS'98 - Pittsburgh - USA - maio 1998 (aceito para publicação)

Anexo A : MAF.idl

```
typedef sequence<string> Stringseq;
typedef sequence<octet> Byteseq;

interface Callback {
        oneway void message (in string agid, in Byteseq agmsg );
};

interface MAF1 {

        string open_agent (in string agent, in string marker, in string myhost, in Stringseq
itinerary, in string home, in boolean kill);

        string get_ref ( in string agref);
        oneway void send_msg (in string agid, in Byteseq msg);
        boolean lock (in string agref);
        oneway void unlock (in string agref);
        void new_ref (in string agref, in string newref);
        oneway void set_history (in string agref, in Byteseq memory);
        Byteseq get_history (in string agref);
        oneway void set_killer (in string agref);
        oneway void set_end(in string agref);
        void set_clientref(in Callback cref);
};
interface Agent {

        attribute Stringseq itinerary;
        readonly attribute Byteseq memory;

        void run ();
        Byteseq save (out string next_host, out boolean end);
        void set_Home (in string home);
        void set_memory (in string value, in long index);
};
interface MAF2 {

        boolean receive_agent (in string myhost, in string home, in string initialref, in string
agserv, in Byteseq state, in Stringseq itinerary, in string marker);

        void register (in string myhost, in string home, in string initialref, in string agserv, in
Byteseq state, in Stringseq itinerary, in string marker, in string servclass, in string agclass, in
Byteseq serverimpl, in Byteseq agimpl);
};
```