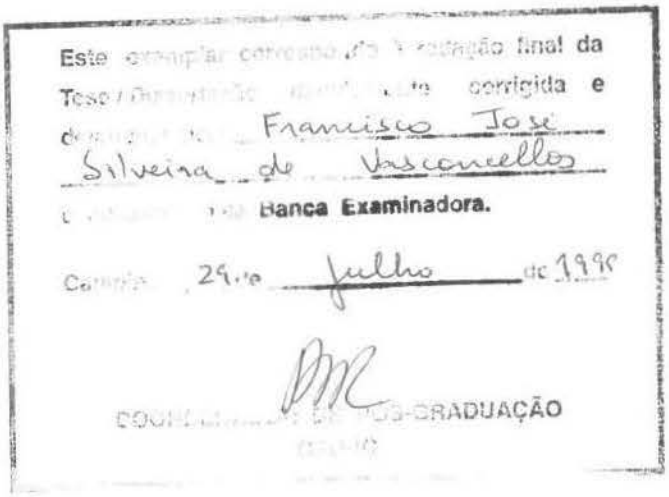

Projeto e desenvolvimento de um suporte a agentes móveis para a plataforma Multiware

Francisco José Silveira de Vasconcellos

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Projeto e Desenvolvimento de um Suporte a Agentes Móveis para a Plataforma Multiware

Francisco José Silveira de Vasconcellos

Abril de 1999

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)
  Instituto de Computação — UNICAMP
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação — UNICAMP
- Dr. Maurício de Menezes Cordeiro
  Instituto de Pesquisas da Marinha — IPqM
- Prof. Dra. Maria Beatriz Felgar de Toledo (Suplente)
  Instituto de Computação — UNICAMP

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas, como requisito parcial para a obtenção do título de mestre em Ciência da Computação

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Vasconcellos, Francisco José Silveira de

V441p Projeto e desenvolvimento de um suporte a agentes móveis para a plataforma Multiware / Francisco José Silveira de Vasconcellos -- Campinas, [S.P. :s.n.], 1999.

Orientador : Edmundo Roberto Mauro Madeira

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Programação orientada a objetos (Computação). 2. Redes de computação. 3. Software - Desenvolvimento. I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Projeto e Desenvolvimento de um Suporte a Agentes Móveis para a Plataforma Multiware

Este exemplar corresponde a redação final da dissertação devidamente corrigida e defendida por Francisco José Silveira de Vasconcellos e aprovada pela Banca Examinadora.

Campinas, de Maio de 1999.

Edmundo R Mauro Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 09 de abril de 1999, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Maurício de Menezes Cordeiro
IPqM - Marinha

Prof. Dr. Luiz Eduardo Buzato
IC- UNICAMP

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP

À memória de meu pai
Francisco Athayde de Vasconcellos.

Agradecimentos

A Deus, por estar sempre presente em minha vida.

Ao Prof. Edmundo, pela paciência, orientação, apoio e sobretudo, pela amizade.

A meus pais pela vida e a minha madrasta Maria Stella pelo carinho, dedicação e ensinamentos.

A Jucele, esposa, amiga e companheira, por sua sabedoria e paciência.

A Patrícia, minha "baixinha", por ser motivação para todo e qualquer desafio.

A Marinha do Brasil, instituição de valores nobres e que veio complementar a formação de meu caráter.

Ao Vice-Almirante Odilon Luiz Wollstein pelo incentivo e apoio quando de minha opção.

Ao amigo Augusto Queiroz, sua esposa Mônica e filhos, que me proporcionaram uma acolhida das mais fraternas em sua casa. "Quem encontra um amigo, encontra um tesouro de valor incalculável".

A Alexandre Oliva pela eterna disposição em ajudar e pelas sugestões sempre trazendo uma solução para os obstáculos ora encontrados na implementação.

A Nuccio Zuquello, Paulo Pagliusi, Bruno Schulze e Patrícia Ropelatto que, com seus conselhos ou opiniões, muito colaboraram para a execução deste trabalho.

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 ao comércio eletrônico, computação móvel, gerência remota de recursos, *data mining*, entre outros.

O principal objetivo deste trabalho é descrever a modelagem e implementação de um Suporte a Agentes Móveis para a plataforma Multiware em desenvolvimento na Universidade Estadual de Campinas e que se baseia em um *broker* que segue a especificação CORBA. Assim, os conceitos relacionados aos agentes e à arquitetura do OMG são apresentados estabelecendo as vantagens de se aliar as funcionalidades oferecidas pelo *broker* à capacidade de migração de objetos.

Abstract

Mobile Agents Systems have received much attention lately. Agents with mobility, representing applications and end-users remotely, 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 commerce, mobile computing, remote management of resources, data mining, among others.

The major goal of this work is to describe the modeling and implementation of a Mobile Agent Support to the Multiware Platform developed at State University of Campinas, based on a *broker* CORBA-compliant. Thus, the concepts related to agents and the OMG's Architecture are presented, establishing the advantages to associate the ORB functionalities with the object migration capability.

Conteúdo:

Lista de Figuras

Lista de Siglas

AB	Architecture Board
ANSA	Advanced Networked Systems Architecture
API	Application Programming Interface
ARA	Agents for Remote Action
CORBA	Common Object Request Broker Architecture
CSCW	Computer Supported Cooperative Work
DAI	Distributed Artificial Intelligence
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
DTC	Domain Technology Committee
GIOP	General Inter-ORB Protocol
IA	Inteligência Artificial
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union - Telecommunication Standard Sector
JDBC	Java Database Connectivity
JRMP	Java Remote Method Protocol
JTS	Java Transactions Service
MASIF	Mobile Agent System Interoperability Facility
NFS	Network File System
OMA	Object Management Architecture
OMG	Object Management Group

ORB	Object Request Broker
PAS	Publicy Available Specifications
POA	Portable Object Adapter
PTC	Platform Technology Committee
RFP	Request for Proposal
RMI	Remote Method Invocation
RM-ODP	Reference Model for Open Distributed Processing
RPC	Remote Procedure Call
SIG	Special Interest Groups
TF	Task Force
vcc	Virtual Code Compiler

Capítulo 1

Introdução

A popularização da Internet, devido a quantidade de informações disponíveis a um custo relativamente baixo, provocou um crescimento exponencial do número de usuários. A "grande rede" e seus problemas passaram a fazer parte da vida das pessoas e das empresas. Muitas pesquisas se iniciaram e, a cada dia, surgem soluções para problemas, novas aplicações são propostas e, conseqüentemente, novos problemas aparecem e são objetos de mais pesquisas. Atualmente fazemos compras, nos comunicamos e até mesmo trabalhamos sem nos afastarmos de nossas residências. Assim também empresas oferecem seus produtos e serviços, fecham negócios e estabelecem ou migram suas redes administrativas para a Internet, configurando assim, suas intranets e extranets.

Juntamente com o aumento da distribuição da informação, a tecnologia aplicada aos microprocessadores e aos sistemas de comunicação utilizados para a conexão de computadores também evoluem de modo acelerado. Cada vez temos microprocessadores mais velozes e tecnologias que oferecem maior rapidez e

flexibilidade à interconexão das redes locais com as redes de longo alcance. Neste novo e complexo mundo da informação, a centralização dá lugar à distribuição pois, permitindo o compartilhamento de informações e recursos, aumentamos a disponibilidade, a confiabilidade e a performance, obtendo uma melhor relação entre custo e desempenho [Tanenbaum, 1996].

Apesar das vantagens citadas, a distribuição também se depara com vários problemas e, em especial, com a heterogeneidade. Novas tecnologias estão sempre surgindo e a integração das novas soluções às existentes muitas vezes é impossível. O custo de se abrir mão de um investimento feito anteriormente, e que, ainda não se pagou pelo uso, faz com que muitas organizações mantenham, com novos investimentos em tecnologias mais recentes, uma larga variedade de equipamentos, incluindo PCs, *workstations*, minicomputadores e *mainframes*. Estes equipamentos, por sua vez, rodam diferentes sistemas operacionais e se encontram, na maioria das vezes, em diferentes arquiteturas de redes.

Visando resolver o problema da heterogeneidade e da distribuição, vários desenvolvedores passaram a oferecer serviços que operavam com protocolos e interfaces de programação padrões [Bernstein, 1996]. Estes serviços foram chamados de serviços de ***middleware*** por se situarem no meio (*middle*), ou seja, acima da camada do sistema operacional e *software* de rede, e abaixo das aplicações.

Segundo [Rymer, 1996], a escolha do *middleware* passou a ser talvez mais importante que a dos sistemas operacionais visto que não mais se trata apenas de um *software* de comunicação, mas de um ambiente de execução que pode agregar ainda ferramentas de desenvolvimento. O *middleware* provê serviços que conectam interfaces de usuários, aplicações, dados e outras partes. Segundo [Benda, 1997], o coração do *middleware* é o serviço de comunicação inter-processos e cita, como um dos sistemas de mais adequados, a arquitetura desenvolvida pelo ***Object***

Management Group (OMG), denominada Arquitetura de Gerência de Objetos[1] (OMA). A OMA, orientada a objetos, possui sua base em um negociador de requisições denominado ***Object Request Broker*** (ORB), especificado através da ***Common Object Request Broker Architecture*** (CORBA), que atua como um barramento de objetos. Os objetos que se ligam ao *broker* são classificados pela OMA de acordo com os serviços por eles oferecidos.

Os conceitos ligados a objetos como encapsulamento, reusabilidade, abstração, portabilidade e modularidade, além do fato das interações entre os objetos serem tratadas através de mensagens, indicam naturalmente sua utilização em sistemas distribuídos [Vinoski, 1997]. A abordagem de computação de objetos distribuídos vem se expandindo rapidamente, provocando um esforço de padronização tanto pela ISO[2] e ITU-T[3] através da elaboração de um Modelo de Referência para o Processamento Distribuído Aberto - RM-ODP[4] [ISO, 1995a], [ISO, 1995b] e [ISO, 1995c], quanto pelo OMG. Os conceitos básicos para a criação do RM-ODP nasceram com uma outra plataforma orientada a objetos chamada ANSA (*Advanced Networked Systems Architecture*) e a plataforma do OMG surgia como um esforço independente. Atualmente podemos notar que, tanto o desenvolvimento da OMA segue conceitos estabelecidos pelo Modelo de Referência da ISO, quanto o RM-ODP tem sido atualizado por serviços melhor especificados pelo OMG. Podemos concluir que a diferença básica entre as duas especificações está no nível de abstração do enfoque. Esta conclusão é reforçada pela requisição do OMG junto à ISO para tornar-se um *Submitter of Publicy Available Specifications* (PAS). Segundo [Gage, 1997] pelo fato do OMG ser um consórcio sem fins lucrativos, a aprovação era esperada após cerca de seis meses. Assim, já a partir de

---

[1] *Object Management Architecture*

[2] *International Organization for Standardization*

[3] *International Telecommunication Union - Telecommunication Standard Sector*

[4] *Reference Model for Open Distributed Processing*

1999, o OMG poderá submeter a especificação CORBA e outras partes da OMA como padrão internacional ISO. A certificação ISO é muitas vezes um requisito nas compras de tecnologias em vários países.

A linguagem de programação **Java**, orientada a objetos e com várias bibliotecas prontas e voltadas para o desenvolvimento de aplicações distribuídas, é muitas vezes apontada como uma forte concorrente de CORBA. O impacto de Java no mundo da computação está fora de questionamentos e pode ser facilmente comprovado pela quantidade de material publicado, seja em livros ou em congressos, que trata da linguagem. A portabilidade de código, independência de plataforma de execução e a fácil integração do código com os *browsers* tornam Java um meio quase ideal de desenvolvimento para redes. Motivados com o fenômeno causado pela linguagem, a Sun Microsystems vem desenvolvendo diversos projetos baseados em Java como aconteceu com o *Java Remote Method Invocation*, atualmente já incorporado ao *kit* de desenvolvimento e, mais recentemente, com uma arquitetura que, como um ORB, oferece diversos serviços, denominada *Enterprise Java*. Ainda assim, não podemos apontar Java e os produtos associados como a solução dos problemas citados anteriormente. Embora Java ofereça a independência de plataformas, não se trata de uma tecnologia integradora como CORBA. No transcorrer deste trabalho poderemos ver que as plataformas, ditas concorrentes, são na verdade complementares e discorreremos também sobre os possíveis concorrentes de CORBA, comparando-os a esta arquitetura.

Além do problema da heterogeneidade, o crescimento do uso da Internet e, dentro das empresas, o aumento da distribuição dos dados e aplicações, vêm causando problemas de sobrecarga de tráfego e dificuldades na coleta e processamento de dados remotos. Mesmo com os avanços das tecnologias de comunicação de dados este problema permanece pois cada vez mais cresce o número de usuários conectados e cada vez mais as informações e dados estão globalmente distribuídos e de uma maneira não estruturada. Isto nos leva ao

paradoxo de termos uma quantidade enorme de informações que se tornam muitas vezes inatingíveis devido aos problemas citados.

A solução para este tipo de problema parece estar em um mecanismo, inicialmente pesquisado na área de Inteligência Artificial, conhecido como **agentes**. Inteligentes ou não, uma classe de agentes despertou o interesse dos pesquisadores da área de redes e sistemas distribuídos pois apresentam características que indicam uma solução mais adequada a alguns problemas da área. Esta classe de agentes é denominada **agentes móveis** e vem sendo objeto de diversas pesquisas e implementações. A utilidade deste mecanismo é comentada em [Chess et al., 1994] e revista em [Chess et al., 1997].

O OMG, dentro da OMA, define uma estrutura que serve de suporte para agentes móveis em um ORB. Embora haja previsto tal suporte, este não foi especificado como esperado.

O presente trabalho foi iniciado exatamente com os requisitos estabelecidos pelo OMG para a referida especificação, visando prover ao ORB comercial OrbixWeb, desenvolvido pela Iona Technologies e atualmente em uso no Instituto de Computação como o *middleware* do projeto de pesquisa da plataforma **Multiware** [Mendes et Madeira, 1994], um mecanismo de suporte a agentes móveis, contribuindo assim, para o desenvolvimento de futuras aplicações utilizando-se este paradigma. Tais agentes, situados em um ORB, além de se beneficiarem de sua natureza migratória e da independência de plataformas oferecida por Java, poderão interagir com outros objetos estáticos e desenvolvidos em qualquer das linguagens suportadas por CORBA.

O paradigma de agentes móveis trata-se de um mecanismo para desenvolvimento de aplicações não somente para a Internet, mas também para WANs e LANs, onde se busque na migração do código, soluções para problemas como balanceamento de carga ou otimizações de aplicações distribuídas como coleta em grandes bancos de dados.

1.1 Estrutura da dissertação

No capítulo 2, são expostos os conceitos sobre a estrutura do OMG e sua Arquitetura de Gerência de Objetos. São apresentadas também as funcionalidades mais relevantes da linguagem de programação Java, estabelecendo as comparações pertinentes e a análise de como estes dois "mundos" se completam. Por fim é apresentada a Plataforma Multiware.

O capítulo 3 cobre os conceitos relacionados aos agentes, mais especificamente aos agentes móveis, e apresenta alguns sistemas de agentes móveis. A idéia inicial do OMG para especificar o suporte a agentes móveis para um ORB é analisada e como se encontram os trabalhos relacionados a esta especificação é comentado. É apresentada também uma outra especificação relacionada ao tema conhecida como ***Object by Value***.

No capítulo 4, a proposta deste trabalho é detalhada, estabelecendo-se as diferenças em relação aos sistemas de agentes móveis existentes e também em relação ao enfoque atualmente dado para a especificação do OMG para o referido suporte.

No capítulo 5 são apresentados a modelagem da implementação e os resultados obtidos. Este trabalho é então concluído com o capítulo 6 onde são apresentadas as contribuições e sugestões para futuras pesquisas.

Capítulo 2

Corba, Java e a Multiware

Neste capítulo descrevemos em maiores detalhes aspectos referentes à sistemática de especificação do OMG e sua arquitetura, bem como apresentamos a evolução de Java como plataforma de desenvolvimento de aplicações distribuídas, demonstrando como estas plataformas podem se completar. Por fim, descrevemos sucintamente a Plataforma Multiware.

2.1 OMG e CORBA

Embora muito já exista escrito sobre CORBA, muito também se confunde como se esta fosse a Arquitetura básica do OMG. Nesta seção apresentaremos a estrutura organizacional do OMG, sua Arquitetura de Gerência de Objetos e, naturalmente, CORBA e a Arquitetura de Facilidades Comuns.

2.1.1 OMG e sua estrutura

O *Object Management Group* é o maior consórcio composto por empresas ligadas à informática, contando já com mais de 800 membros [URL 04]. Trata-se de

uma companhia, sem fins lucrativos, que foi criada em maio de 1989, inicialmente com 8 membros: 3Com Corporation, American Airlines, Canon, Data General, Hewlett-Packard, Philips Telecommunications, Sun Microsystems e Unisys Corporation [Yang et Duddy, 1996].

A organização não desenvolve nenhuma tecnologia. Ela provê uma estrutura para que seus membros especifiquem tecnologias e produzam implementações comerciais que sigam estas especificações.

O processo do OMG enfatiza a cooperação, o compromisso e o acordo final ao invés da escolha de uma das soluções possíveis propostas por um dos membros [Vogel et Duddy, 1997]. Todas as propostas de especificação passam por um processo de análise e posterior votação pelos membros.

A missão do OMG é promover a teoria e a prática de tecnologia de objetos para o desenvolvimento de sistemas computacionais distribuídos e seu objetivo principal é o desenvolvimento de uma arquitetura comum, independente de plataforma de hardware e de sistemas operacionais, para a intercomunicação de objetos [Yang et Duddy, 1996].

Esta arquitetura é freqüentemente confundida e tratada como CORBA. CORBA é, na verdade, a base da arquitetura OMA e assim apenas uma parte desta que, em seu todo, procura abranger todos os aspectos da computação em um ambiente distribuído. A OMA e a CORBA serão descritas mais adiante.

O esforço de desenvolvimento desta tecnologia é feito por todos os membros. Uma notável exceção é a Microsoft que, embora membro do OMG, desenvolve uma tecnologia concorrente e proprietária conhecida como **DCOM** (*Distributed Component Object Model*).

A estrutura organizacional do OMG

A estrutura organizacional do OMG, como mostrada na Figura 2.1, é composta de um conselho[5] que administra a organização e ratifica as atividades dos outros grupos do OMG. A maioria dos cargos são ocupados por voluntários, representantes das companhias-membro, e que não recebem salários.

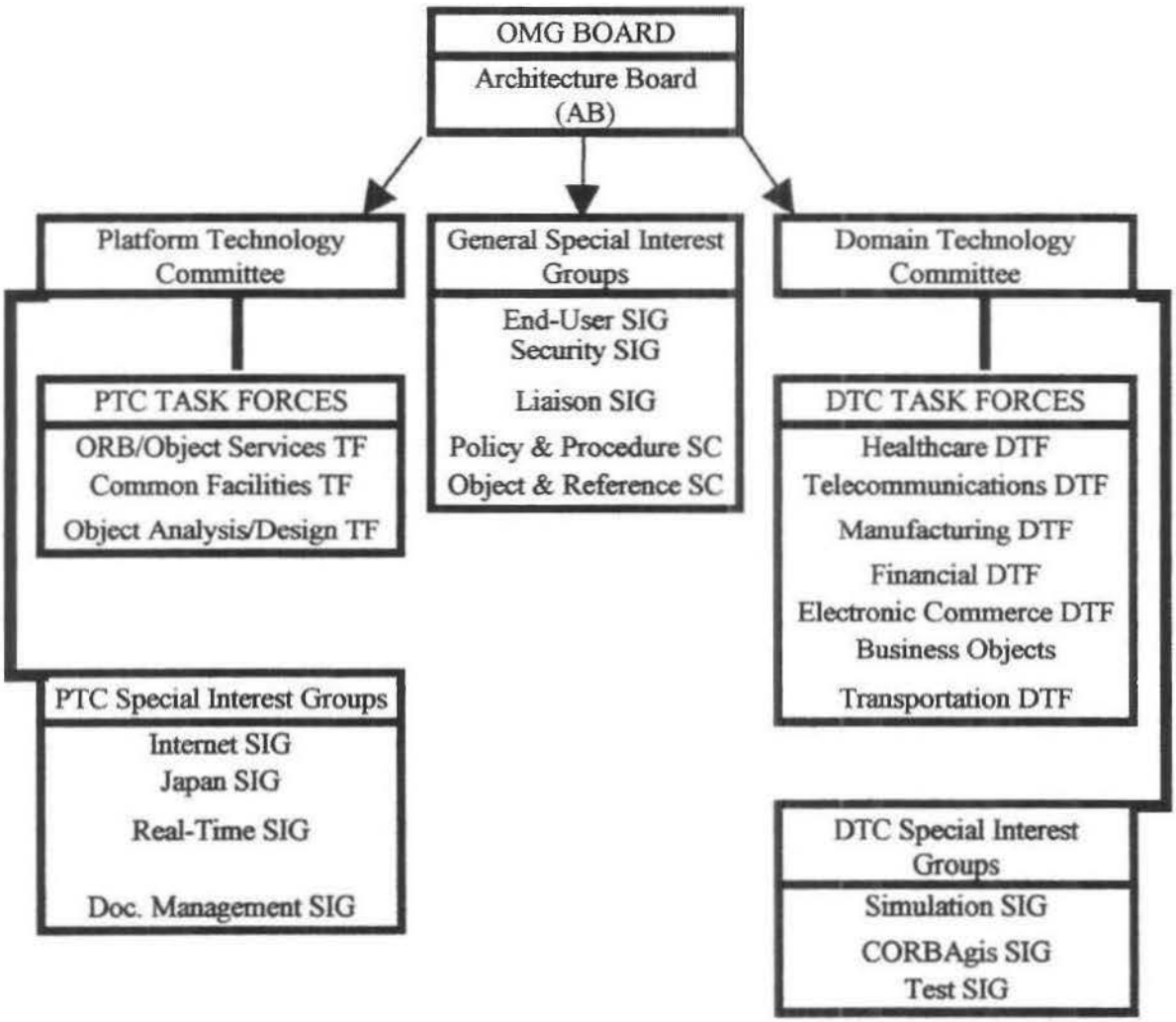

Figura 2.1 - Organização do OMG [Vogel et Duddy, 1997].

[5] OMG *Board*

Os grupos técnicos são supervisionados pelo Conselho de Arquitetura[6] (AB) cujos membros são projetistas de sistemas experientes. O AB é eleito pelos membros do OMG. Ele revê todas as propostas de tecnologias e especificações submetidas para verificar consistência e conformidade com a Arquitetura de Gerência de Objetos (OMA). A estrutura dos comitês, forças-tarefas e outros grupos refletem a OMA como poderemos comprovar em seguida. Dois comitês supervisionam a adoção de tecnologias de um número de forças-tarefas (TF) e grupos de interesse especiais (SIGs).

O Comitê de Tecnologia da Plataforma (PTC) se preocupa com a infra-estrutura da OMA: o *Object Request Broker* (ORB), as Facilidades Comuns e Serviços de Objetos; bem como com o relacionamento da OMA com o projeto e análise orientados a objetos.

O Comitê de Tecnologia de Domínios (DTC) se preocupa com tecnologias que suportem o desenvolvimento de aplicações específicas como no caso de manufaturas, comércio eletrônico, medicina ou telecomunicações.

As Forças-Tarefas podem emitir *Requests for Proposals* (RFPs) que são descrições de problemas a serem resolvidos. As respostas são solicitadas na forma de especificações na Linguagem de Definição de Interfaces (IDL[7]) com a semântica explicada em inglês. Duas coletas de submissões são realizadas, geralmente espaçadas de três meses, e então uma especificação é selecionada por voto dos membros e apresentada ao Comitê que controla a Força-Tarefa. Os submetedores devem implementar a tecnologia cuja especificação submeteram de modo a, caso sua proposta venha a ser aceita pelos membros do OMG, tornarem disponível comercialmente no prazo de um ano.

---

[6] *Architecture Board*

[7] *Interface Definition Language*

Os Grupos de Interesse Especiais (SIGs) não podem emitir RFPs diretamente ou adotar especificações tecnológicas. Porém podem fazê-lo com o suporte de alguma Força-Tarefa.

2.1.2 A OMA

A Arquitetura de Gerência de Objetos (OMA), mostrada na Figura 2.2, estabelece uma estrutura com termos e definições para especificações de interfaces de objetos em um ambiente distribuído.

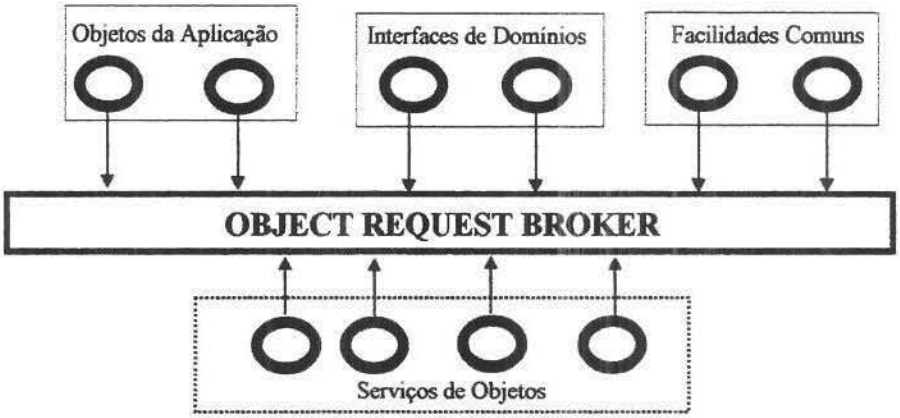

Figura 2.2 - Arquitetura de Gerência de Objetos do OMG.

- 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 [OMG formal/98-02-33].
- Os **Serviços de Objetos** (*CORBAServices*) definem uma coleção de serviços 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***). Assim, através da IDL, estes componentes 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 (Figura 2.3).

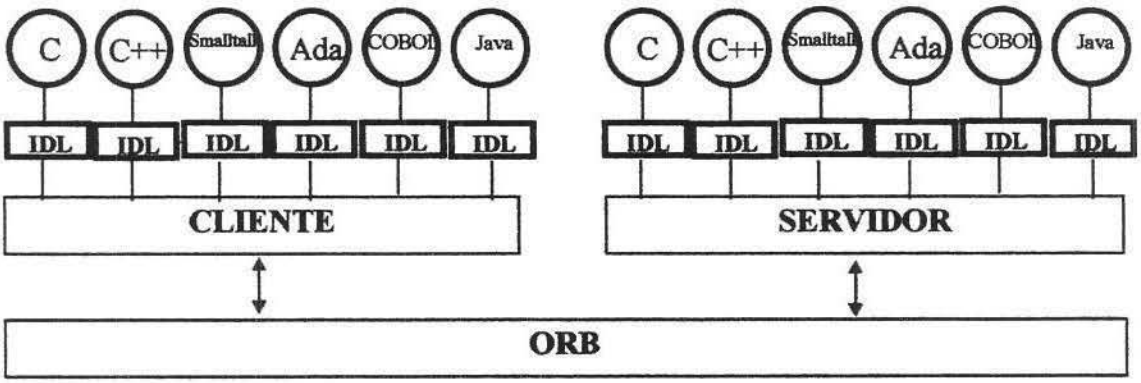

Figura 2.3 - Interfaces IDL provêem interoperabilidade cliente/servidor[Orfali et al., 1996].

2.1.3 CORBA

A plataforma descrita pela CORBA é composta pelo núcleo do ORB e de suas interfaces, permitindo a interação transparente entre os clientes e as implementações de objetos, como são conhecidos os provedores de serviços (Figura 2.4). 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,** que é uma entidade que possui a informação necessária para identificar um objeto em um ORB. É utilizada em invocações de métodos para localizar um Objeto CORBA. Uma referência de objeto identifica o mesmo Objeto CORBA sempre que usada. É possível, entretanto, que múltiplas referências de objetos se refiram a apenas um Objeto CORBA.

O **CLIENTE** é qualquer código que invoca uma operação em um Objeto CORBA, independente da linguagem de implementação, da plataforma de execução e de sua localização. Um cliente pode ser ou não um Objeto CORBA, ou mesmo pode ser um programa não orientado a objeto.

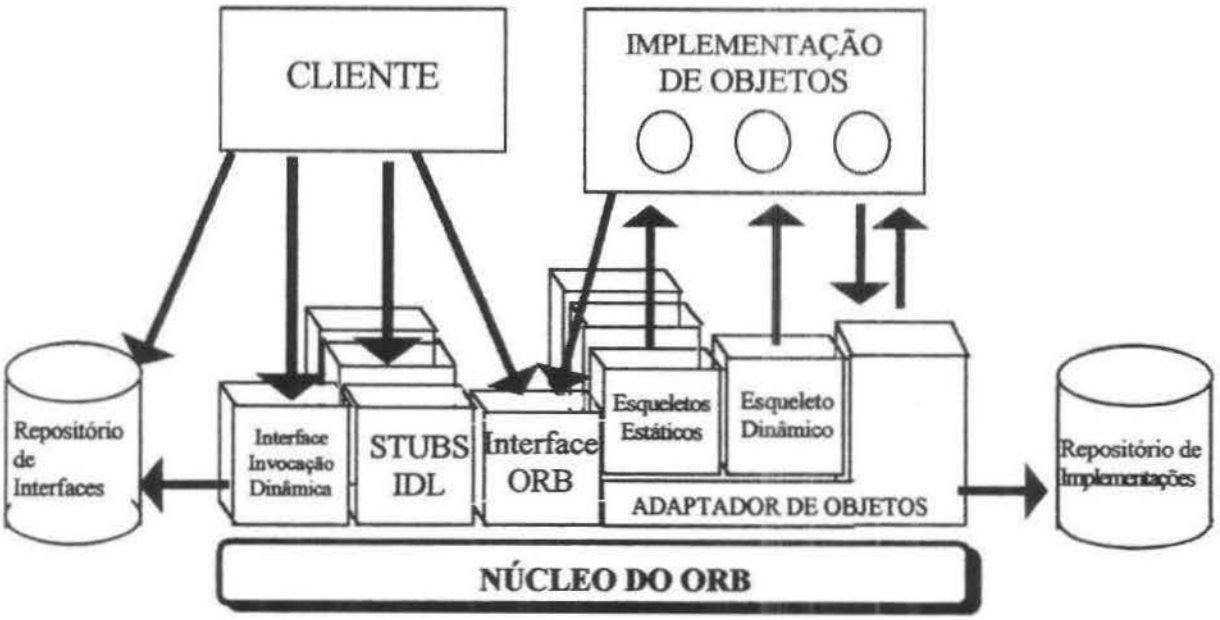

Figura 2.4 - Estrutura do ORB[OMG formal/98-02-33].

O **Núcleo do ORB** é responsável por receber as requisições dos clientes através da interface estática (*IDL-STUB*) ou da **Interface de Invocação Dinâmica** e localizar a implementação de objetos apropriada, transmitindo os parâmetros necessários, transferindo o controle através de uma interface **Esqueleto-IDL** (*IDL Skeleton*) **estática** ou **dinâmica** e retornando as respostas, caso existam.

Os **STUBs**, gerados pelo compilador IDL, contêm código para executar a serialização, isto significa que codifica a operação e seus parâmetros para formatos de mensagens que poderão ser enviadas ao servidor.

A **Interface de Invocação Dinâmica (DII[8])** permite a um cliente descobrir, em tempo de execução, o método a ser invocado. O cliente especifica o objeto a ser invocado e obtém a descrição do método, através de uma seqüência de chamadas, no **Repositório de Interfaces** que possui os métodos e parâmetros de todas as interfaces registradas.

Situado sobre o núcleo do ORB, interligando serviços e aceitando requisições de serviços das implementações de objetos está o **Adaptador de Objetos**. A especificação diz que qualquer ORB precisa prover um adaptador de objetos padrão denominado **POA** (*Portable Object Adapter*) e que servidores poderão suportar mais de um adaptador de objetos. Ele provê um ambiente para ativar objetos, passar requisições para estes e atribuir referências de objetos. O adaptador de objetos também registra as classes que suporta e os objetos no **Repositório de Implementações**.

O ORB possui dois repositórios. O **Repositório de Implementação** contém informações que permitem localizar e ativar implementações de objetos e o **Repositório de Interfaces** provê o armazenamento consistente de definições de interfaces.

---

[8] *Dynamic Invocation Interface*

O **Esqueleto Estático** provê interfaces para cada serviço exportado pelo servidor. Assim como os STUBs, é criado através do compilador IDL.

A **Interface de Esqueleto Dinâmica (DSI[9])** é um mecanismo para a entrega de requisições para uma implementação de objetos quando esta não é conhecida em tempo de compilação. Esta Interface é também usada para implementar pontes entre ORBs e também pode ser usada por interpretadores e linguagens *script* para dinamicamente gerar implementações de objetos.

2.1.4 Arquitetura de Facilidades Comuns

A Arquitetura de Facilidades Comuns preenche o espaço conceitual entre os Serviços de Objetos e Objetos de Aplicação, podendo ser especializações dos Serviços de Objetos. As Facilidades Comuns ou *CORBAFacilities* oferecem serviços de nível mais elevado, amplamente utilizáveis por várias aplicações.

As Facilidades Comuns (Figura 2.5) eram divididas em duas categorias principais: **Facilidades Horizontais** e **Verticais** [OMG formal/97-06-15].

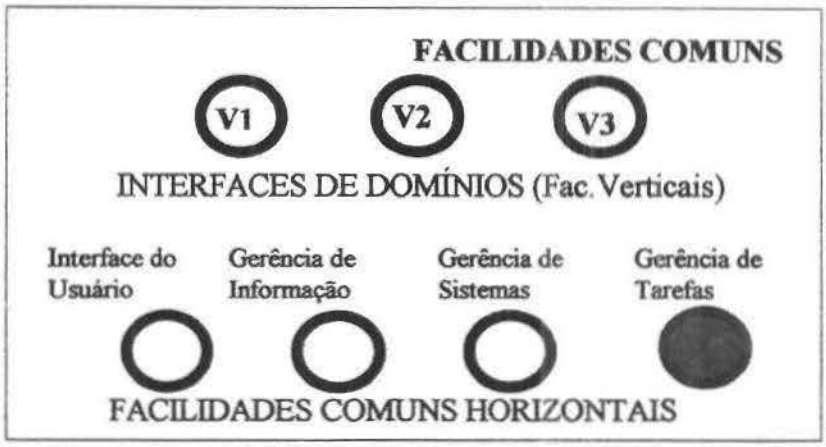

Figura 2.5 - Facilidades Comuns

[9] *Dynamic Skeleton Interface*

As Facilidades Verticais, que oferecem serviços a áreas especializadas como Processamento de Imagem, Manufatura e Indústria de Exploração de petróleo, passaram a ser tratadas como **Interfaces de Domínios** (Figura 2.2), embora a especificação ainda não tenha sido alterada.

As Facilidades Horizontais, onde se enquadra o assunto desta proposta, caracterizam-se por funções compartilhadas pela maioria dos sistemas, independente da área de aplicação, e são subdivididas em quatro tipos:

- INTERFACE DE USUÁRIO
  - permite o acesso do usuário ao sistema de informações.
- GERÊNCIA DE INFORMAÇÃO
  - cobre a modelagem, definição, armazenamento, recuperação, gerência e intercâmbio da informação.
- GERÊNCIA DE SISTEMA
  - permite a gerência de sistemas de informação.
- GERÊNCIA DE TAREFA
  - habilita a automação de trabalho, tanto de processos de usuários como processos de sistemas pertencentes a um mesmo sistema de informação.

O presente trabalho se concentra nas Facilidades Comuns Horizontais de Gerência de Tarefa, que são as seguintes:

- Automação - permite acesso à funcionalidade chave de um objeto a partir de outro.
- Gerência de Regras - suporta a aquisição de conhecimento, manutenção e execução de objetos baseados em regras, como agentes inteligentes.
- *Workflow* - provê gerência e coordenação de objetos que são parte de um processo de trabalho.
- Agentes - provê suporte para agentes estáticos e dinâmicos.

2.1.5 O futuro

Em um recente documento do OMG [OMG orbos/97-05-25], IBM, Netscape, Oracle e Sunsoft apresentam a necessidade de, não somente CORBA, mas toda a OMA, tornar mais simples o desenvolvimento de Objetos CORBA e buscar maior integração com a World Wide Web.

Neste sentido, é feita uma referência a Java e apontam o caminho para a definição de componentes CORBA compatíveis com *JavaBeans*. Um ***JavaBean*** é um componente de software reutilizável que pode ser manipulado visualmente em uma ferramenta de construção [Souza, 1998].

Na seção seguinte abordaremos Java e a infra-estrutura usada para a comunicação e transporte de componentes Java.

2.2 Java

A linguagem de programação Java foi bem recebida pela comunidade mundial de desenvolvedores de software. Estes podem se beneficiar do desenvolvimento, apenas uma vez, da codificação de aplicações, sem a preocupação com a necessidade de "portar" suas aplicações para diversas plataformas de software e hardware [Arnold et Gosling, 1996].

Para muitos, Java é uma ferramenta para a criação dos *applets*. ***Applet*** é o termo usado para uma mini-aplicação escrita em Java e executada dentro de uma página Web. Java é, de fato, valiosa para ambientes de rede distribuídos como a Web. Contudo, Java vai além, sendo uma linguagem de programação de propósito geral, adequada a construção de uma série de aplicações que não dependem das características da rede.

Nascida como linguagem de programação, Java vem crescendo de forma a se tornar uma plataforma completa. Em paralelo ao *kit* de desenvolvimento 1.0, era lançada a estrutura do ***Remote Method Invocation*** (RMI) que foi posteriormente integrado ao *kit* de desenvolvimento 1.1. O RMI possibilita não somente a invocação

remota de objetos, daí sua natural comparação com CORBA, mas também a passagem de objetos para execução remota, preservando o estado de execução (passagem por valor) através do ***Object Serialization***.

2.2.1 Java RMI e Object Serialization

O RMI foi desenvolvido para suportar a invocação de métodos em objetos remotos, ou seja, em outras máquinas virtuais Java. Como em CORBA, os clientes RMI interagem com os objetos remotos através de suas interfaces que, por serem definidas em Java, dispensam a utilização de uma IDL.

O RMI utiliza o Java *Remote Method Protocol* (JRMP) e o *Object Serialization* para introduzir a característica que vem sendo utilizada pelos sistemas de agentes móveis e que provocou uma busca de adaptação, por parte do OMG, de sua arquitetura: a passagem de objetos por valor. A passagem do estado de um objeto em execução é possível em Java através do ***Object Serialization***. A passagem das classes que implementam estes objetos é otimizada pelo fato de ser somente efetuada a passagem das classes inexistentes no destino do objeto. Quando um objeto vai ser recuperado no destino, é feito o *download* somente das classes necessárias.

A serialização de Java captura o estado do objeto, armazenando os valores de seus campos e, se outros objetos são referenciados, monta o grafo de dependências respectivo. Tudo isto é colocado em um objeto do tipo *OutputStream* e recuperado através de um objeto *InputStream*. Para tal, existem diferentes extensões destes objetos básicos no *package* java.io do JDK 1.1 [Flanagan, 1997]. Porém um cuidado deve ser tomado pois nem todas as classes Java podem ser serializadas. Somente classes que implementem a interface *Serializable* ou a interface *Externalizable* podem ser escritas ou recuperadas de um *stream*.

O *Object Serialization* é utilizado por todos os sistemas de agentes móveis baseados em Java conhecidos por nós.

2.3 CORBA e Java

Como vimos, o RMI pode ser comparado a um *broker*, porém não possui os serviços oferecidos pela OMA. Com o desenvolvimento da ***API Enterprise Java*** (incluindo RMI, *Java Database Connectivity-JDBC, Java Naming, Java Transactions Service-JTS*), o que nasceu como linguagem de programação se transforma em uma plataforma para aplicações distribuídas.

A comparação com a OMA torna-se inevitável e através de diversos artigos em revistas especializadas, tanto o OMG quanto a Sunsoft tentam demonstrar o caminho da integração. Em verdade, o concorrente comum é a plataforma de objetos distribuídos proprietária da Microsoft denominada ***Distributed Component Object Model (DCOM).***

Não é o escopo deste trabalho a comparação entre estas tecnologias, mas o leitor interessado encontrará comparações entre CORBA e DCOM em [Chung et al., 1997], [Orfali et Harkey, 1997a], [Orfali et Harkey, 1997b], [Resnick, 1997] e [Baker, 1997].

2.3.1 O que Java oferece à CORBA

Inegavelmente Java vem provocando modificações no paradigma de desenvolvimento de aplicações cliente/servidor. A simplicidade da linguagem, a portabilidade de código, a grande quantidade de bibliotecas prontas para o desenvolvimento de aplicações distribuídas e o crescimento constante da plataforma vêm fazendo com que as empresas busquem se aliar a esta nova tecnologia.

Uma característica em especial vem provocando mudanças profundas na arquitetura do OMG. A serialização de objetos, que permite a passagem de objetos por valor, veio provocar a emissão, por parte do OMG, do ***Object by Value Request for Proposal*** [OMG orbos/96-06-14]. Até então, um *broker* só permitia a passagem de objetos por referência. Como este fato está intimamente relacionado ao presente trabalho, será abordado com maior detalhe no próximo capítulo.

Embora o OMG vise a independência de linguagem de programação, o sucesso de Java vem provocando a emissão de documentos visando uma maior e mais rápida integração desta linguagem. Assim temos o ***Java-to-IDL Request for Proposal*** [OMG orbos/97-03-08] que visa propiciar aos desenvolvedores Java integrarem suas aplicações ao ORB sem a necessidade de definirem suas interfaces IDL.

2.3.2 O que CORBA oferece à Java

CORBA oferece à Java a possibilidade de se integrar a aplicações desenvolvidas em outras linguagens através do uso de IDL. Além disto possui uma arquiteutra estruturada de serviços, seguindo um padrão de desenvolvimento aberto. Sabe-se que o Serviço de Transações Java (JTS) foi baseado no Serviço de Transações da OMA [Curtis, 1997].

Por fim, o uso de um *broker* fornece as transparências de acesso e localização e, utilizando-se o *Internet Inter-ORB Protocol* (IIOP), a interoperabilidade entre diferentes ORBs.

2.3.3 IIOP e RMI

Assim como o OMG envidou esforços na direção de Java, a Sunsoft em seu mais recente *kit* de desenvolvimento Java (JDK 1.2) inclui um ORB que, além de suportar o mapeamento Java-IDL, possui um Serviço de Nomes baseado no Serviço de Nomes da OMA [Orfali et al., 1997].

O RMI seria então implementado não mais utilizando-se o Java *Remote Method Protocol* (JRMP) e sim IIOP. Isto não é inteiramente certo. A Sunsoft, embora tenha incluído a capacidade de usar diretamente o IIOP, não abandonou o JRMP e também não demonstra pretender fazê-lo.

Em [Orfali et Harkey,1997a], no capítulo 12, os autores comparam a performance do uso de IIOP versus RMI (entenda-se JRMP) e apresentam resultados demonstrando ser o IIOP mais eficiente.

2.4 A Plataforma Multiware

A plataforma Multiware [Loyolla et al., 1994; Mendes et Madeira,1994], provê uma estrutura que possibilita o trabalho cooperativo em ambientes de serviços abertos, utilizando-se dos conceitos do Modelo de Referência para Processamento Distribuído Aberto (RM-ODP) da ISO.

Essa plataforma é composta de três camadas (Figura 2.6):

- Hardware / Software básico
- Middleware
- Groupware

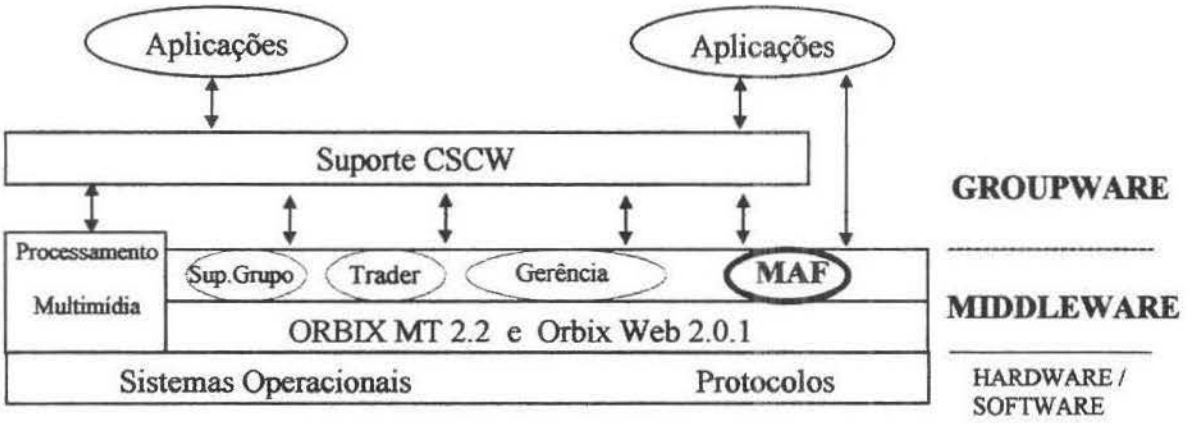

Figura 2.6 - Plataforma Multiware

A camada Hardware / Software básico é composta por sistemas operacionais e por protocolos de comunicação.

A camada Middleware é responsável por prover as facilidades de processamento distribuído aberto para a camada Groupware e para as aplicações. É composta por :

- **Processamento Multimídia** que possibilita a troca de informações multimídia em tempo real com uma qualidade de serviço especificada;

- por um ORB e;
- por uma subcamada que provê novas funcionalidades a esse ORB. O suporte a mobilidade, escopo deste trabalho, é um exemplo de uma nova funcionalidade.

A camada Groupware oferece funcionalidades a classes diferentes de aplicações como CSCW (*Computer Supported Cooperative Work*) e DAI (*Distributed Artificial Intelligence*), entre outras.

Esta plataforma vem sendo desenvolvida na Universidade Estadual de Campinas, conjuntamente pelo Instituto de Computação e pela Faculdade de Engenharia Elétrica e de Computação. Assim, temos um ambiente de desenvolvimento distribuído em redes distintas. Em ambos laboratórios são utilizados os ORBs comerciais da Iona Technologies. Atualmente estão instaladas e em uso a Orbix 2.2 Multi-Thread (com mapeamento IDL para C++) e a OrbixWeb 2.0.1 (com mapeamento IDL para Java). A nova versão do ORB com mapeamento Java (OrbixWeb 3) está instalada no Instituto de Computação e se encontra em avaliação para um futuro *upgrade*.

Capítulo 3

Agentes Móveis em CORBA

O esforço do OMG, pretendendo especificar sua Facilidade de Agentes Móveis, demonstra seu objetivo em manter sua arquitetura atualizada com as pesquisas realizadas em torno de sistemas distribuídos. Nesta seção apresentamos os conceitos relacionados ao paradigma dos agentes móveis e descrevemos o estágio atual do trabalho do OMG referente ao tema.

3.1 Agentes e agentes móveis

3.1.1 Agentes

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

Mesmo em uma área específica, como IA (Inteligência Artificial) por exemplo, há controvérsias. Em [Franklin et Graesser, 1996] encontramos um apanhado de definições e uma tentativa de se classificar e definir agentes. Esta

tentativa foi comentada e criticada em [Castelfranchi, 1996], [Petrie, 1996] e [Wooldridge,1996] sem que tenha sido estabelecido um consenso a respeito.

Segundo [Nwana et Wooldridge, 1997] 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 [Wooldridge, 1996] 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.

Como parte da edição de julho/agosto de 1997, a revista IEEE Internet Computing [URL 09] promoveu uma mesa redonda onde renomados pesquisadores responderam a questões relacionadas a agentes. Uma das questões foi exatamente se é prejudicial esta confusão acerca da definição de agente. Embora muitos considerarem negativa a confusão, alguns compararam o fato com o que ocorreu no fim dos anos 80 ao surgir o termo objeto. Além disto, consideram natural este tipo de problema quando diferentes áreas de pesquisa atuam sobre um mesmo tema, cada qual buscando o seu enfoque.

Em geral, todos temos uma noção do que seria um agente e as definições deste termo em dicionários de diferentes línguas seguem um senso comum. Em [Ferreira, 1986] encontramos entre outras :

- aquele que opera, agencia, age;
- procurador, delegado, administrador;
- pessoa que trata de negócio por conta alheia.

De acordo com [Green et al., 1997], existe este senso comum no conceito, como um representante que age em benefício de outros. Assim é estabelecida a primeira propriedade fundamental:

- Agir em benefício de outra entidade.

Os mesmos autores definem como segunda propriedade fundamental a autonomia e, a seguir, a pró-atividade[10] e reatividade[11].

Finalmente estabelecem que os agentes podem possuir outros atributos, citando alguns considerados chave como aprendizagem, cooperação e mobilidade.

Assim, a definição poderia ser sumarizada 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".

Evidentemente outros autores definem outros atributos e nos parece acertada a opinião de John Ousterhout, engenheiro da Sun Microsystems e autor da linguagem Tcl e Tk toolkit, quando diz: "Debater sobre terminologia é perda de tempo... Tente somente ser o mais específico que puder (usando a terminologia que quiser) quando estiver falando sobre agentes".

Aplicações para agentes

Outra questão levantada na mesa redonda citada anteriormente foi a respeito da utilidade dos agentes. A pergunta era se existia algum problema que agentes, e somente agentes, poderiam resolver.

---

[10] tradução de *proactivity* - A ausência de pró-atividade seria passividade. O pró-ativo não age somente em resposta ao ambiente.

[11] tradução de *reactivity* - Capacidade de perceber mudanças no ambiente e reagir de modos diversos.

Os pesquisadores mais voltados para IA apontaram a sobrecarga de informações como um problema específico que agentes resolveriam. Em relação aos agentes móveis, classificam-nos como uma solução em busca de um problema. Porém Danny Lange, criador do sistema Aglet da IBM e hoje diretor da divisão de agentes da General Magic, questiona: "Existe algum problema que objetos, e somente objetos, podem resolver? Não, mas muitos preferem objetos pela melhor abstração, produtividade e por uma solução mais apropriada. Não procure um problema que só possa ser resolvido por agentes - esta é uma visão muito simplista do mundo".

Segundo [Green et al., 1997], 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 (o problema da sobrecarga de informações citado acima);
- gradualmente mais usuários são inexperientes;
- conseqüentemente os usuários requerem agentes para ajudá-los a entender o mundo complexo que estamos criando.

3.1.2 Agentes Móveis

Um tipo específico de agentes, denominado usualmente como agentes móveis ou ainda agentes itinerantes [Chess et al., 1995] ou agentes transportáveis [Kotay et Kotz,1994], 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 [Gray et al., 1996] 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 IA [Green et al., 1997] 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 [Petrie, 1997]. O autor, renomado pesquisador de IA, 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 [Berbers et al., 1996], quais sejam:

- autonomia;
- comunicação;
- mobilidade;
- auto-iniciação[12];
- orientado a objetivo.

Nas seções subseqüentes analisamos as vantagens e possíveis aplicações, linguagens utilizadas, infra-estrutura necessária, requisitos de segurança e, finalmente, algumas implementações existentes 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.

---

[12] tradução de *self-starting*

Aplicações para agentes móveis

Analisamos 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 que este paradigma, em determinadas circunstâncias, pode apresentar são:

- eficiência - consomem menos recurso de rede já que migramos a computação até os dados;
- redução de tráfego na rede - visto que a maioria dos protocolos de comunicação envolvem várias interações, especialmente quando medidas de segurança estão habilitadas, com o uso dos agentes temos estas interações acontecendo localmente;
- autonomia assíncrona - tarefas podem ser dadas a agentes e estes operarão assíncrona e independentemente do program que os enviou;
- 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 uma 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;

⇒ Monitorização de informações remotas;

⇒ Processamento paralelo;

⇒ Gerência de redes e de sistemas distribuídos;

⇒ Entretenimento (jogos);

⇒ Computação móvel.

Em [Muller, 1996] o autor aborda o uso de agentes inteligentes para gerência de sistemas e em [Lingnau et Drobnik, 1995] os agentes móveis são apontados como uma possível solução para a gerência de grandes redes, além de outros problemas como a recuperação de informações e o comércio eletrônico [Milojicic et al., 1996].

O comércio eletrônico, sem dúvida, tem sido uma das áreas de maior pesquisa para agentes móveis e é citado também em [Merz et Lamersdorf , 1996].

Outra área na qual agentes móveis parecem ser solução para diversos problemas é a computação móvel e isto é citado em [Lingnau et Drobnik, 1995], [Gray et al., 1996] e também em [Chess et al., 1995].

Assim como nos parece insensato procurar um problema específico o qual somente agentes poderiam resolver, também o é não reconhecer que este paradigma trará muitos benefícios e soluções melhores aos problemas das áreas citadas e de muitos mais que ainda irão surgir.

Linguagens Utilizadas

Na teoria, qualquer linguagem pode ser usada. Existem dois tipos de soluções para execução de agentes móveis: compilação dinâmica e interpretação [Berbers et al., 1996] [Lingnau et Drobnik, 1995].

- Na primeira, o código de um agente é transportado na forma de código fonte que será dinamicamente compilado em cada local para onde o agente se mover;
- No caso de linguagens que são diretamente interpretadas como Tcl ou que são transportadas através de um código intermediário como Java, há a necessidade de se ter um interpretador apropriado em cada local.

As vantagens dos agentes móveis têm motivado o surgimento de vários trabalhos relacionados ao desenvolvimento de sistemas que explorem e otimizem o uso destes. Java e Safe Tcl podem ser usados como componentes em sistemas de agentes e existem sistemas como Tacoma, Agente Tcl, Telescript entre outros, desenvolvidos especificamente para suportar agentes móveis.

Infra-estrutura para agentes móveis

As características dos agentes móveis, anteriormente citadas, determinam a infra-estrutura necessária para seu suporte. Por isso, cabe aqui revê-las:

• **Autonomia**	- são capazes de tomar iniciativas e exercer o controle sobre suas ações.
• **Comunicabilidade**	- estão aptos a comunicar-se com outros agentes ou com serviços disponíveis.
• **Mobilidade**	- são capazes de se transportar de uma máquina para outra sobre diferentes plataformas e sistemas.
• **Auto-iniciação**	- diferente de programas padrões que são diretamente invocados pelo usuário, um agente pode perceber mudanças em seu ambiente e decidir quando agir.
• **Orientado a objetivo**	- um agente aceita requisições de alto nível indicando o que o cliente quer, e é responsável por decidir como e onde satisfazê-lo.

Os agentes precisam de uma infra-estrutura que os suporte e auxilie na execução de suas tarefas. Tais ambientes são chamados **agências**. As tarefas básicas de uma agência podem ser resumidas em:

1. Prover facilidades para agentes executarem, como alocar recursos e possibilitar o comportamento autônomo. Uma agência precisa suportar a linguagem de programação do agente, possuindo, conforme o caso, o interpretador ou o compilador apropriado. Se este, ou qualquer outro recurso essencial não estiver disponível, a agência deverá emitir uma mensagem de erro rejeitando ou redirecionando o agente para outra agência. Além disto, é necessário que seja efetuado o registro do agente, para permitir sua localização e interação com outros agentes.
2. Suportar a comunicação dos agentes com outros agentes. Visando propiciar um sistema aberto, mecanismos padrões como *Remote Procedure Call* (RPC) ou invocação de objetos precisam ser suportados. Soma-se a isto a possibilidade do agente interagir com os resultados provenientes desta comunicação. A comunicação assíncrona, no caso do outro agente não estar continuamente conectado, é bastante apropriada. Na maioria dos casos, a comunicação do agente com a origem é essencial.
3. Prover facilidades para o transporte dos agentes em um ambiente heterogêneo. Mover um objeto envolve mover seu código e seu estado (inclusive dados). Parte do estado é dependente da plataforma (registradores, endereços na pilha) o que torna difícil o transporte em uma rede heterogênea. Enquanto sistemas tradicionais, que oferecem esta possibilidade, decidem quando e para onde será feito o transporte, no caso de agentes é o próprio agente que decidirá se mover. Um agente ao decidir se mover, pode limpar seu estado antes de sair de modo a evitar transferir estados dependentes da plataforma. Assim as variáveis de estado que serão transportadas conterão apenas os resultados acumulados e o conhecimento

do agente. A decisão para onde se mover poderá ser simplesmente tomada através de uma rota pré-estabelecida ou ser tomada de forma dinâmica, baseada nos serviços e recursos necessários ou na interação com o ambiente ou com a comunicação com o meio externo. Quando um agente decidir se mover, diferentes partes da agência precisam ser informadas para que as ações apropriadas sejam tomadas, como o preparo do agente para transporte, a retirada do respectivo registro da agência e a liberação dos recursos que estavam sendo utilizados.

4. Prover segurança dos agentes e de suas interações. Desde a recepção do agente é importante verificar se existem problemas relacionados à origem do agente ou às condições do mesmo ao ser recebido (código suspeito ou confuso). A agência poderá estabelecer um controle de acesso a recursos apropriado e, por outro lado, prover mecanismos para evitar erros de interpretação ou possíveis modificações no código do agente.

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[13]) 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 [Milojicic, 1997] 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 e os agentes por sua vez, devem ter protegidos tanto seu código

[13] tradução de *owner*.

quanto seus dados contra ataques efetuados por outros agentes ou por agências não confiáveis.

3.1.3 Sistemas de agentes móveis

Existem vários trabalhos de pesquisa em universidades e também vários sistemas comerciais para suporte a agentes móveis. Devido ao grande número dos sistemas desenvolvidos em Java, vamos relacioná-los em separado.

Sistemas não baseados em Java

Sem dúvida alguma o mais importante, senão o precursor dos sistemas de agentes móveis desenvolvido antes do surgimento de Java foi o **Tabriz** da General Magic, mais conhecido pela linguagem utilizada para os agentes, **Telescript**. Foi o sistema de agentes móveis mais adotado por muitos anos. Com o advento de Java, a própria General Magic abandonou o projeto, dedicando-se ao desenvolvimento do Odyssey [URL 15], inteiramente voltado para agentes em Java.

Entre os trabalhos acadêmicos, merecem destaque o **Agent Tcl** [Gray, 1995] desenvolvido pelo Dartmouth College [URL30], o **ARA** (*Agents for Remote Action*) da Universidade de Kaiserslautern [URL 11] e o projeto **TACOMA** [URL 12] conduzido em colaboração pela Universidade Cornell e pela Universidade de Tromso na Noruega, todos para agentes escritos em Tcl.

Sistemas baseados em Java

Java veio fortalecer e expandir o paradigma de agentes móveis. As características oferecidas por esta linguagem como o RMI, *Object Serialization*, *multithreading*, reflexão computacional, portabilidade entre plataformas e coleta de lixo automática, permitem o desenvolvimento mais natural dos sistemas de agentes móveis [Kiniry et Zimmerman, 1997].

No meio acadêmico, um sistema bastante difundido é o **Mole** da Universidade de Stuttgart [Straßer et al., 1996].

Entre os sistemas comerciais mais conhecidos estão o **Odyssey** da General Magic [URL 15], o **Aglets** da IBM [URL 13] e o **Voyager** da ObjectSpace [URL 07].

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 em HTTP e URLs. Dos sistemas citados, o que nos parece mais flexível é o Odyssey. 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.

O Voyager utiliza a serialização e a reflexão e, através de um compilador denominado *vcc* (*Virtual Code Compiler*), permite a criação de um "objeto virtual" que atua como um proxy para um objeto remoto ou agente. O *vcc* pode ser usado em qualquer classe Java (desenvolvida a partir do JDK 1.1) para criar o objeto virtual. A documentação disponível não fornece maiores detalhes sobre o controle da migração dos agentes e trata o Voyager como um *broker* para agentes, propondo ser melhor que CORBA por não requerer o uso de IDL e assim, ser de desenvolvimento mais fácil. Isto enquanto estratégia de marketing demonstra, no mínimo, total desconhecimento da plataforma do OMG. Se Voyager usa a estrutura de Java, como também o fazem os outros sistemas de agentes Java, não pode ser comparado a um ORB visto que não oferece a independência de linguagens de programação impossibilitando o relacionamento dos agentes com objetos desenvolvidos em outras linguagens.

3.2 A Facilidade de Agentes Móveis

No final de 1995, o OMG emitiu um *Request for Proposal* [OMG cf /95-11-03] de modo a especificar a Facilidade de Agentes Móveis prevista como uma das Facilidades Comuns de Gerência de Tarefas da OMA.

3.2.1 Requisitos

De acordo com o estabelecido pelo OMG no *Request for Proposal* que trata da Facilidade de Agentes Móveis, os requisitos básicos que um modelo deverá suportar são os seguintes:

1. *Marshalling* e *un-Marshalling* de agentes. A Facilidade deverá estabelecer uma maneira de transmitir os pacotes codificados.
2. Codificação e decodificação de *containers* para transporte de agentes. A codificação deverá obedecer ao padrão estabelecido pela linguagem IDL e deverá prover informação necessária para a correta decodificação e execução do agente.
3. Mover agentes de uma máquina para outra.
4. Registro e invocação de agências.
5. Interrogação, pelos agentes, sobre serviços que podem ser fornecidos.
6. Segurança, envolvendo acesso e execução.

Ainda segundo este documento, a Facilidade de Agentes pode opcionalmente prover:

- Identificação e localização de agentes.
- Inicialização, parada e suspensão de execução de agentes.
- Descoberta e monitorização, por outras agências ou outros agentes, de agentes pelo nome, por suas características ou por suas capacidades.

A execução dos agentes e sua interação com o ambiente, envolvendo também o registro e outras funções relacionadas ao ciclo de vida (inicialização, parada, suspensão e novas decisões de se mover), fazem com que a Facilidade de Agentes necessite de serviços de NOME e CICLO DE VIDA adequados. A localização de

agentes e descoberta de outros agentes ou outras agências poderá ser provida pelo serviço de *Trading*.

Com as diferentes abordagens apresentadas em [OMG cf /96-08-01] e [OMG cf /98-08-02], 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 (MASIF[14]) [OMG cf /97-06-04]. Com isto, a submissão final [OMG orbos/97-10-05] tornou-se genérica a ponto de não tratar os agentes como Objetos CORBA e não definir mecanismos de transporte dos agentes por um ORB.

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 [Milojicic et al., 1998], o motivo alegado para não prover mecanismos de codificação para o transporte de agentes foi exatamente a independência de linguagens de programação.

3.3 A Passagem de Objetos por Valor

Em 1996, já influenciado pelo crescimento de Java, o OMG emitiu um *Request for Proposal* [OMG orbos/96-06-14] com o claro objetivo de permitir a passagem de Objetos CORBA por valor. Em CORBA os objetos são passados por referência e, em certas situações, operar sobre a referência de um objeto remoto não é muito eficiente.

3.3.1 Requisitos

Os seguintes requisitos foram estabelecidos para as submissões:

[14] *Mobile Agent System Interoperability Facility*

1. Definir precisamente a semântica da passagem de objetos por valor, especificando o relacionamento entre as identidades e implementações dos objetos no contexto de envio e de recebimento;
2. Descrever como a interoperabilidade entre ORBs diferentes será assegurada;
3. Especificar regras de gerenciamento de memória;
4. Justificar qualquer nova interface IDL ou qualquer modificação nas já existentes;
5. Especificar qualquer modificação que venha a ser necessária no mapeamento das linguagens para IDL;
6. Especificar qualquer modificação necessária ao *General Inter-ORB Protocol* (GIOP) ou na arquitetura de interoperabilidade de CORBA2.

A submissão conjunta adotada [OMG orbos/98-01-18] envolve a adição de uma nova estrutura[15] na IDL do OMG, denominada ***valuetype***. O *valuetype* possui características de *struct* e de interface pois suporta tanto dados quanto operações. Como interfaces, pode derivar de outros *valuetypes.*

Quando um *valuetype* é passado como argumento para uma operação remota, é na verdade o estado de um objeto sendo enviado e que, no recebedor, será instanciado outro objeto com identidade separada da original. Assim, todas as operações invocadas são sempre locais ao processo onde o *valuetype* existe. Naturalmente são definidos mecanismos para que o recebedor localize corretamente a implementação do objeto.

A especificação, introduz o conceito de interface abstrata, o que é estranho visto que as interfaces IDL sempre foram abstratas desde o início de CORBA. Porém, o que mais chama a atenção é a ausência do objetivo estabelecido no *Request for Proposal*, qual seja a passagem de Objetos CORBA por valor. Embora a

[15] Tradução de *construct*

especificação permita que *valuetypes* tornem-se Objetos CORBA por herança de interfaces IDL, somente as referências de objetos poderão ser passadas como argumentos neste caso. *Valuetypes* não são Objetos CORBA e sim um novo tipo IDL. Passar Objetos CORBA por valor é um problema difícil devido às interações com adaptadores de objetos, às questões relacionadas a localização de objetos, identidade dos objetos e quando estes podem ser copiados e movidos e às questões relacionadas à migração de objetos entre ORBs de diferentes fabricantes [Vinoski, 1998].

O autor referenciado acima, termina seu artigo lamentando que a especificação *Objects by Value* tenha quebrado uma tradição no processo de adoção de tecnologias do OMG. Segundo ele, este processo sempre foi conservador, largamente baseado em tecnologias bem conhecidas e comprovadas. Atualmente o OMG está revendo os procedimentos com o objetivo de prevenir, no futuro, similares adoções apressadas e desavisadas.

3.4 Trabalhos relacionados

Em [Kotay et Kotz, 1994], é apresentado um sistema para transporte de agentes que utiliza *rsh* (*remote shell*) como mecanismo de transporte, mas não provê nenhum modo de comunicação entre agentes.

Em [Harker,1995], Kenneth Harker estende o trabalho citado anteriormente, descrevendo as desvantagens do mecanismo usado para o transporte, adotando TCP/IP e *sockets* BSD tanto para o transporte como para implementar a comunicação entre agentes escritos em Tcl.

Outra implementação é encontrada em [Lingnau et al., 1995] e [Lingnau et Drobnik,1995] que, com um método específico de empacotamento dos agentes (*MIME Internet standard*), usa o HTTP para o transporte.

Existem muitos sistemas comerciais e projetos acadêmicos de suporte a agentes móveis. Porém, embora muitos citem CORBA em determinados aspectos

somente o sistema MAGNA, em desenvolvimento pelo GMD-Fokus [Krause et al., 1997], trata os agentes como Objetos CORBA.

Os principais sistemas de agentes já foram citados no capítulo anterior e a principal diferença do nosso MAF para todos os outros sistemas está na integração do suporte com o *broker*, 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, ser possível a utilização de um ORB para o transporte de agentes. De acordo com [Orfali et Harkey, 1997a], o *broker* não perde em eficiência para as outras formas de conexão.

O único sistema realmente relacionado com o presente trabalho é o MAGNA. Na verdade, evoluiu para o sistema **GRASSHOPPER,** já disponível para *download* [URL 03] e que requer o Visibroker para Java.

Este sistema segue o MASIF, permite o transporte de agentes por *sockets*, RMI ou IIOP. Se por um lado isto deixa transparente ao usuário o desenvolvimento de uma IDL, por outro não deixa claro como tratar o mapeamento de referências ao migrar um agente CORBA. Este é um problema que é deixado para o Serviço de Nomes dos ORBs, o que seria a solução teórica mais correta. Na prática porém, nem todos os ORBs possuem implementação do Serviço de Nomes, tornando o Grasshopper totalmente dependente do Visibroker ou da OrbixWeb. Cita ainda a possibilidade de localização de agentes usando um *trader* o que demonstra a possibilidade do agente ter uma referência de objeto, mas não traz uma interface IDL para agentes e sim utiliza um recurso específico, somente do Visibroker, de geração automática de *stubs*.

Assim, de todos os sistemas de agentes, o único que procura realmente se adequar a migração dos agentes em um *broker* é o Grasshopper, tratando inclusive de aspectos não abordados pela especificação do MASIF como o transporte dos agentes através do IIOP. O principal e talvez único problema deste sistema é a sua

dependência do mecanismo de geração automática de *stubs* não padronizado e somente disponível no Visibroker.

Por se tratar de um sistema novo que já demonstra tanta flexibilidade e que, segundo a documentação disponível em [URL 03], aborda também o problema de segurança, deverá se firmar como um dos mais utilizados em futuras pesquisas na área.

Capítulo 4

Modelagem do Suporte a Agentes Móveis

A nossa proposta é o desenvolvimento de um suporte a agentes móveis que será chamado **MAF** por se basear no *Mobile Agent Facility* do OMG. Neste capítulo apresentaremos a idéia geral do trabalho definindo as funcionalidades deste suporte.

A Plataforma Multiware tem como base a CORBA e, para aplicações baseadas em um ORB, como citadas em [Schulze et Madeira, 1997] e [Ropelatto et al., 1998], nosso sistema de suporte a agentes permitirá 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 ao ORB de outra tecnologia de agentes móveis somente para a implementação dos agentes móveis.

4.1 Hipóteses assumidas

O objetivo deste trabalho é o desenvolvimento de um suporte para prover a Facilidade de Agentes Móveis na plataforma Multiware. Tal suporte é baseado, em um ORB para Java atualmente instalado no Instituto de Computação e provê, além das implementações necessárias à mobilidade dos agentes, um serviço de registro de agentes e a possibilidade de comunicação entre agentes.

Para efeito do desenvolvimento das interfaces necessárias ao transporte no núcleo do ORB, é usada a linguagem IDL e considerado o agente escrito em Java.

Estabelecemos como meta a busca por uma solução perfeitamente integrada a CORBA, utilizando o protocolo IIOP para o transporte dos agentes. Os agentes são tratados como Objetos CORBA, usufruindo assim de todos os benefícios advindos do uso desta plataforma.

O próprio suporte ou MAF, como iremos referenciá-lo deste ponto em diante, será um Objeto CORBA de modo a permitir a invocação remota das aplicações e dos agentes.

Inicialmente, pelo conhecimento do funcionamento do *broker*, identificamos alguns obstáculos a serem solucionados:

- A passagem de objetos por valor em CORBA, ou seja, a possibilidade de se capturar o estado de execução de um objeto, enviá-lo para um outro ponto na rede e reiniciá-lo preservando seu estado. Como vimos anteriormente, a passagem de objetos por valor em um ORB, embora especificada, não tratou a passagem por valor de Objetos CORBA, permanecendo possível somente a passagem por referência.
- Como efetuar a identificação de objetos móveis utilizando-se o padrão do OMG que é o uso de referências de objetos. O ORB a ser utilizado já possui um mecanismo de identificação porém foi implementado para ser utilizado com

objetos estáticos pois na referência de objeto é inserido o nome do *host* no qual este foi instanciado.

- Como permitir um grau de autonomia ao agente. Para que o objeto venha a ser considerado um agente móvel, este deverá ter o controle sobre a migração. O *broker* controla todo o ciclo de vida de um objeto e não prevê este grau de autonomia.
- O registro dos agentes. O registro de um objeto no ORB é feito pelo desenvolvedor através de comandos específicos do ORB. Faz-se necessário um mecanismo para que o agente possa se registrar no destino sem a interferência do desenvolvedor.

4.2 A Modelagem

Objetivos

Na fase inicial da 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 itinerário dos agentes é composto por nomes de máquinas e este itinerário pode ser alterado a qualquer momento, tanto pelo próprio agente quanto externamente. Esta funcionalidade, além de proporcionar uma grande flexibilidade aos desenvolvedores de agentes e de aplicações que usam agentes, provê um maior grau de autonomia ao agente móvel.

Podemos notar que, destes objetivos iniciais, advêm várias outras características para a modelagem. Entre elas a necessidade de 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 objeto funciona como um relocador que permite que a localização e a comunicação com os agentes ocorram de modo transparente, independente de onde o agente esteja. Em certos casos, quando for importante o conhecimento preciso, o registro da referência atual permite a localização dos agentes.

Para sistemas sob o NFS[16], o registro dos agentes somente é necessário no domínio do ORB visto que o *broker* utiliza o NFS para instanciar os objetos em qualquer *host* deste. Para o caso de o agente ser enviado para um *host* fora do domínio, no qual não estiver registrado, o suporte provê 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 tema de trabalhos futuros. O enfoque primordial é a migração de Objetos CORBA, porém o modelo não inviabiliza os mecanismos de segurança estudados atualmente.

[16] *Network File System*

A Figura 4.1 apresenta um possível cenário de uma migração de agentes (as setas indicam o sentido da migração).

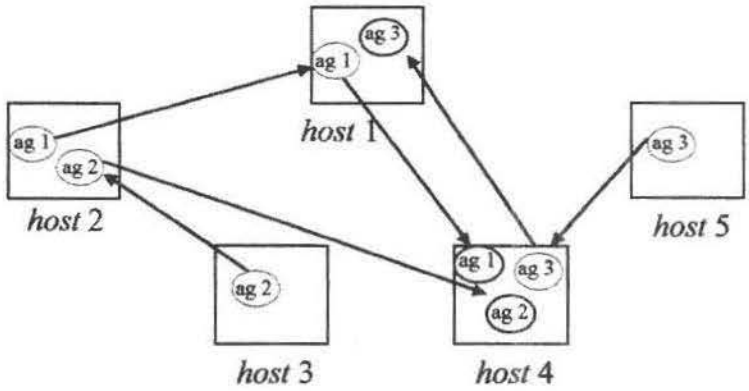

Figura 4.1 - Cenário de uma possível migração de agentes.

Neste cenário, o agente 1 (*ag1*) foi criado no *host* 2, passou pelo *host* 1 e se encontra no *host* 4. Da mesma maneira o *ag2* foi criado no *host* 3, passou pelo *host* 2 e foi para o *host* 4. Por fim, o *ag3* foi criado no *host* 5, passou pelo *host* 4 e se encontra no *host*1. O sistema deve tratar os agentes de forma independente e com mesma prioridade. O *host* 4 pode executar o *ag1*, receber o *ag2* e enviar o *ag3* ao mesmo tempo.

Classes básicas do suporte

O suporte deve ser mais simples quando apenas recebe agentes do que quando instancia um agente atendendo a uma aplicação (cria um agente). Isto porque vislumbramos a necessidade de controlarmos o ciclo de vida dos agentes em seu *host* inicial. Esta decisão de projeto foi necessária pela falta de um Serviço de Nomes que mapeasse as modificações da referência de objeto do agente de modo a permitir sua transparente localização (Um Serviço de Nomes para Objetos Móveis). A utilização do Serviço de Nomes atualmente implementado pelos ORBs não atenderia visto que este serviço é concebido para objetos estáticos e não prevê a mudança de referência de um objeto.

A utilização de um Serviço de Nomes seria benéfica visto que este seria de conhecimento de todos os Objetos no ORB. Este Serviço deveria prover além do mecanismo de mapeamento de referências, a possibilidade de armazenamento de informações de agentes móveis.

Desta forma duas interfaces IDL foram definidas:

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

MAF2 - utilizada para receber agentes já criados em outros *hosts*.

O modelo destas interfaces, de acordo com [Rumbaugh et al. 1991], denominadas respectivamente **MAF1i** e **MAF2i**, é mostrado na Figura 4.2. Também é mostrada a classe **Mthread** que permite a criação de threads de execução independentes para cada agente.

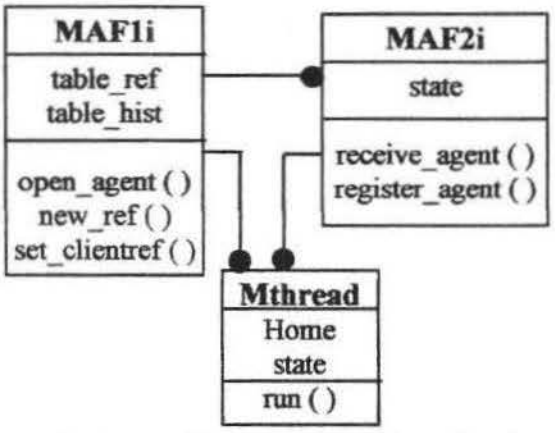

Figura 4.2 - Classes MAF1i, MAF2i e Mthread.

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 os atributos **`Home`** que trata-se de referência de objeto do **MAF1i** que controla este agente e **`state`** para armazenar o estado de execução do agente, além do método *run*, típico de *threads*.

O ciclo de vida de um agente inicia e é controlado pela classe **MAF1i**. O agente será criado pelo método *open_agent*. 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 coletadas pelos 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. O método *new_ref* será explicado em seguida.

Nos *hosts* 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 um para muitos, demonstrada na Figura 4.2, 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 da classe **MAF2i** ser bem mais simples já que nada armazena sobre os agentes. Esta classe possui apenas o atributo **`state`** que é o estado de execução do agente, que será passado para **Mthread**, e os métodos para receber o agente e instanciá-lo (*receive_agent*) e para, caso necessário, registrar o agente no ORB (*register_agent*).

Conforme dito anteriormente, um dos problemas a ser tratado quando migramos Objetos CORBA é que na definição das referências de objetos nos ORBs, 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**. Para tal, foi criada uma tabela de mapeamento de referências de objetos denominada **`table_ref`** que

armazenará, no *host* inicial do agente, a sua referência atual. Além disto, como o agente muda constantemente de referência e esta tabela será freqüentemente atualizada através do método *new_ref* ( ), precisamos criar um mecanismo que evitasse a chamada a um agente que já estivesse 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 temporariamente que o agente se mova caso a **`table_ref`** seja consultada e impedir a consulta desta quando o agente for se mover. Podemos perceber três diferentes componentes do sistema: o **suporte** propriamente dito, o usuário do suporte que seria a **APLICAÇÃO** e o **desenvolvedor de agentes**.

Interface básica para agentes

O suporte irá trabalhar com agentes que, por serem Objetos CORBA, possuem uma interface IDL definida por desenvolvedores de agentes. Visando facilitar o desenvolvimento destes agentes existe uma classe denominada **BaseAgent** que implementará uma interface IDL padrão para agentes e deverá ser especializada pelos desenvolvedores de agentes (Figura 4.3).

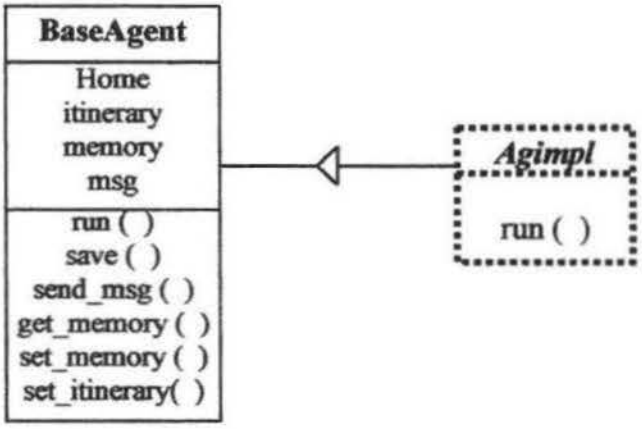

Figura 4.3 - Classes BaseAgent e Agimpl.

A classe específica do agente é mostrada 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 os seguintes atributos:

- **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**: usado para armazenar o itinerário do agente;
- **memory**: atributo genérico o suficiente para armazenar qualquer tipo e quantidade de dados usados ou coletados pelo agente. Será usado para representar o estado de execução e para o armazenamento de um histórico de dados coletados pelos agentes;
- **msg**: também genérico de modo a permitir que qualquer tipo de dado possa ser passado como conteúdo de uma mensagem.

Em **BaseAgent** existem métodos para atuar nos atributos **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 se tornará bastante simples pois a modelagem deixará ao desenvolvedor uma interface básica para agentes com atributos e métodos suficientemente flexíveis de modo a permitir que diversos tipos de agentes sejam criados.

Um exemplo ilustrativo de como um agente seria criado e de como os atributos, que representam seu estado de execução, seriam preservados é mostrado a seguir:

```
package Agentcounter;
import MAF.*;
public class Agcounter_Impl extends BaseAgent {
        int count;

        Agcounter_Impl () {
                super();
                count = 0;
        }
        public void run () {
                for (int i=0;i<15;i++) {
                        count++;
                        try { Thread.sleep(1000); }
                        catch (InterruptedException e) {}
                }
                memory.addElement(_CORBA.Orbix.myHost());
                memory.addElement(new Integer(count));
        }
}
```

Este agente irá, em cada *host* do itinerário, contar 15 segundos. Como o estado é preservado, ele conta até 15 no primeiro *host*, de 16 até 30 no segundo e assim por diante.

Em cada *host* serão colocados no **memory** o nome da máquina e o valor final da contagem nesta máquina. Cabe ressaltar que no exemplo o agente interage com o meio e em cada *host* ele captura o nome do mesmo através do método *myHost*( ). Neste caso específico foi utilizado um método específico de um ORB, mas qualquer outro poderia ter sido usado.

Lançamento de agentes

O suporte deve permitir que agentes sejam lançados por uma APLICAÇÃO, que não necessariamente será um Objeto CORBA, ou diretamente através de

comandos de teclado. Para tal, duas classes foram criadas conforme mostrado na Figura 4.4.

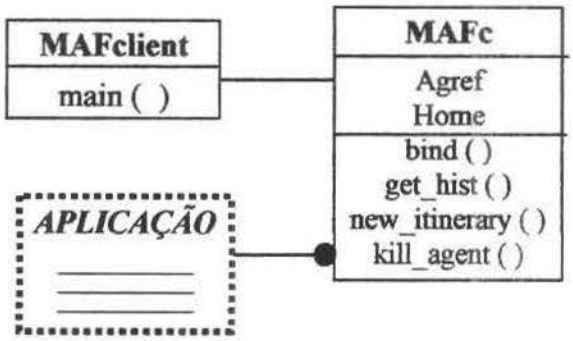

Figura 4.4 - Classes MAFclient e MAFc.

Existirá uma instância de **MAFc** para cada agente lançado. 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 **MAF1i**.

A classe **MAFclient** 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*.

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`**).

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

A APLICAÇÃO aparece em pontilhado pois não é escopo desta modelagem. Enquanto MAFclient só permite o lançamento de um agente, uma APLICAÇÃO poderá lançar quantos agentes forem necessários.

Comunicação com a aplicação

O suporte deve prover meios de comunicação não somente entre Objetos CORBA mas também entre agentes e entre os agentes e aplicações cliente que porventura não sejam Objetos CORBA, ou seja, não estejam registradas no ORB. Para tal, será necessário um mecanismo de *Callback*, que permita ao agente invocar métodos em uma aplicação.

Assim, 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** como mostrado na Figura 4.5.

Callback_abs 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.

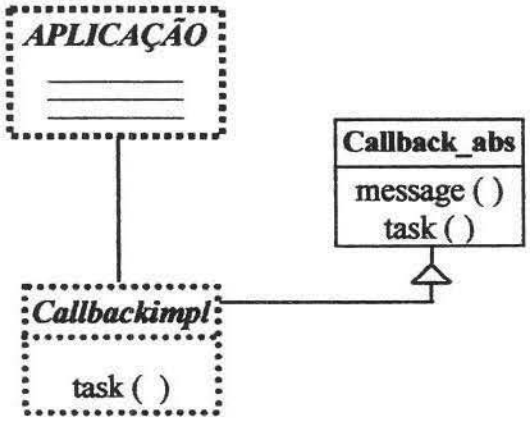

Figura 4.5 - Classes Callbackimpl e Callback_abs

Assim como a APLICAÇÃO, a classe **Callbackimpl** ficará a critério do usuário e por isto ambas aparecem na modelagem em pontilhado.

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.

Modelo proposto

Assim, chegamos ao modelo estático apresentado na Figura 4.6.

A interface IDL do suporte (**MAF.idl**), transcrita no Anexo A, define as interfaces Callback, MAF1, MAF2 e Agent que devem ser então implementadas em uma classe que será especializada para cada tipo de agente.

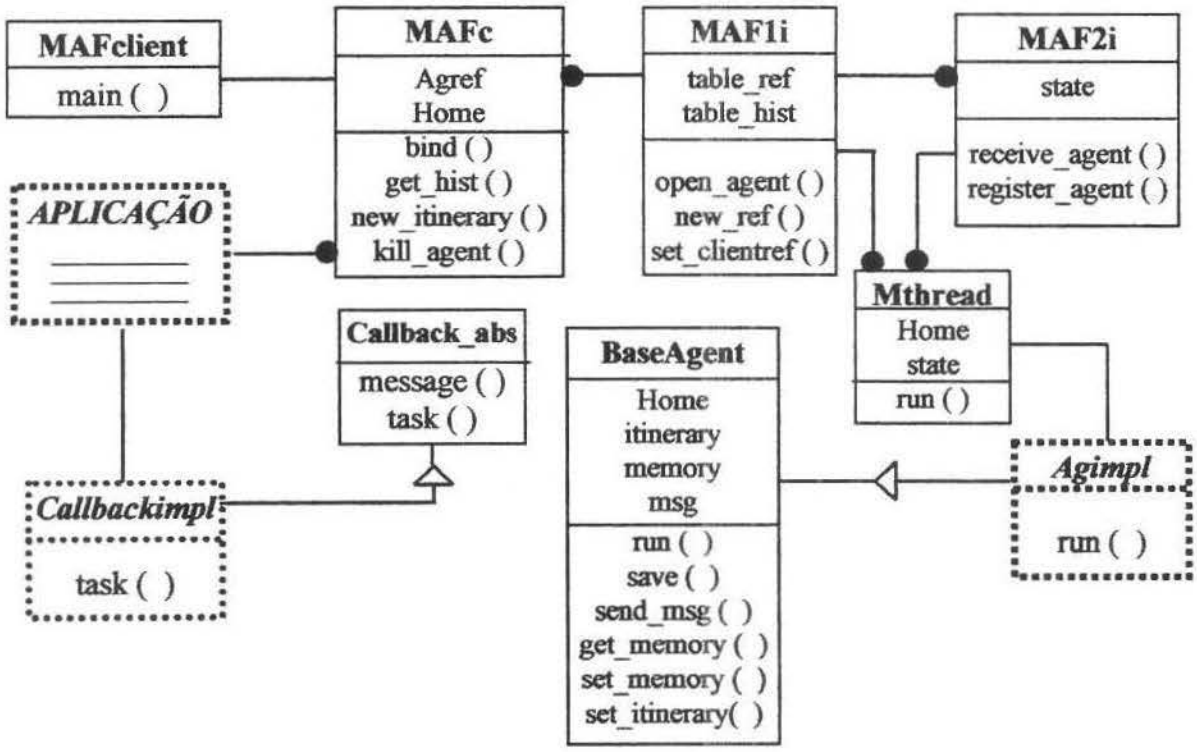

Figura 4.6 - Modelagem do Suporte a Agentes Móveis.

Capítulo 5

Desenvolvimento do MAF

A implementação segue o modelo proposto no Capítulo 4, com todas as classes e respectivas funcionalidades desenvolvidas. Neste capítulo apresentaremos como se deu o desenvolvimento do MAF e as características de uso do suporte.

5.1 Ambiente da 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 é usado o protocolo IIOP.

Tanto na OrbixWeb 2.0.1 quanto na Orbix 2.1, temos já implementado o mecanismo de *Callback* estabelecido como necessário na modelagem [Iona I, 1996].

5.2 Passagem de código e estado

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 [Iona II, 1996]. Como a interface ***Agent*** deve ser genérica optamos por utilizar o gabarito denominado *sequence* em [OMG formal/98-02-33] e implementado neste ORB possibilitando a definição de um tipo sem tamanho definido. Assim, na interface IDL do MAF, 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.

Devido ao NFS, 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 [Iona II, 1996] (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.

5.3 Adaptações ao ORB

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 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, como não é possível a criação de uma classe base para ser especializada, optamos por definir um modelo a ser usado na construção dos servidores. Tal modelo é apresentado no Anexo B e o chamamos ***Agentserver***.

5.4 Exemplo de lançamento de agentes

A seguir, demonstramos o lançamento de dois agentes que, partindo de um mesmo *host*, perfazem itinerários diferentes. 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 conforme mostrado na Figura 5.1.

itinerary1 = {X, A}

itinerary2 = {X, B}

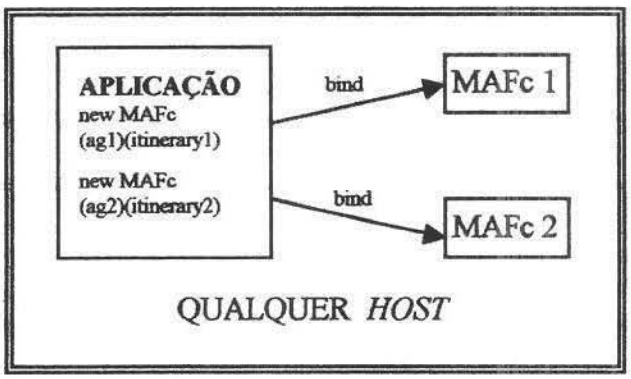

Figura 5.1 - Lançamento de dois agentes.

No *host* inicial foram instanciados dois **MAFc** que executarão um *bind* para os respectivos *hosts* iniciais dos agentes. No caso, ambos efetuarão o *bind* para o *host* X. Será necessário apenas um **MAF1i** visto que os agentes servem a um mesmo "dono".

Após a criação do suporte em X, os **MAFc** invocam o método *open_agent* em **MAF1i** como na Figura 5.2, passando entre outros parâmetros, os agentes a serem criados e os respectivos itinerários.

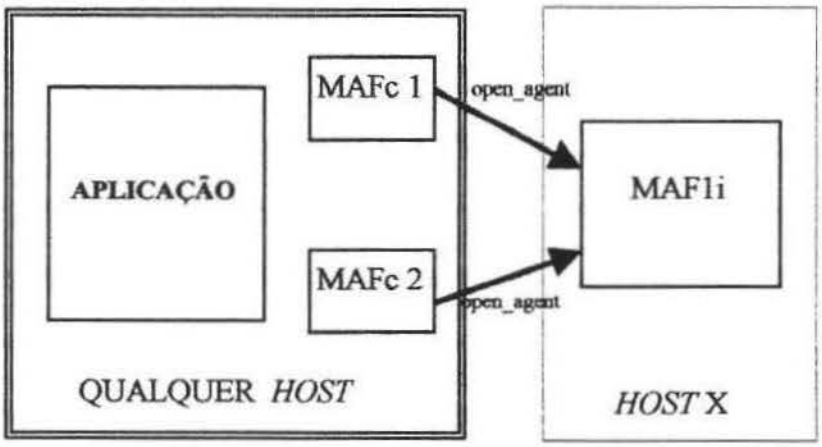

Figura 5.2 - Solicitação de criação dos agentes.

O **MAF1i** ao executar cada *open_agent*, criará um *thread*, através da instância de um **Mthread**, para a execução dos agentes (Figura 5.3). Os agentes serão executados em *threads* de mesma prioridade.

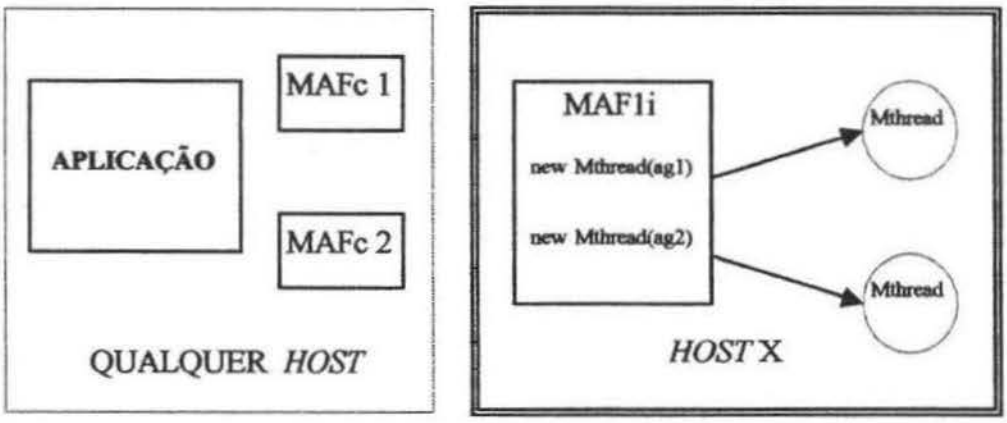

Figura 5.3 - Criação de threads para a execução dos agentes.

Cada **Mthread** se encarregará de inciar a execução dos agentes como mostrado na Figura 5.4.

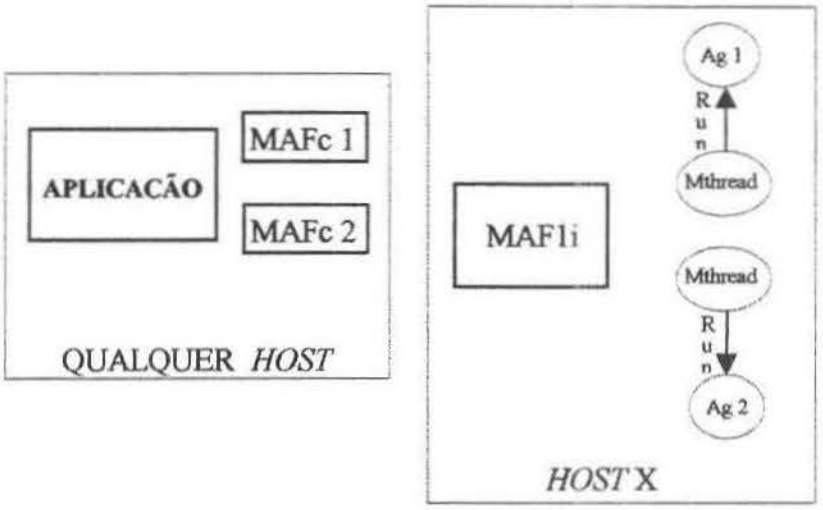

Figura 5.4 - Inicialização dos agentes.

Quando determinado, internamente pelo próprio agente ou externamente por uma chamada no método ***save***, cada **Mthread** suspenderá a execução dos agentes e capturará seu estado de execução (Figura 5.5) e, a seguir invocará um objeto **MAF2i** no *host* seguinte do itinerário.

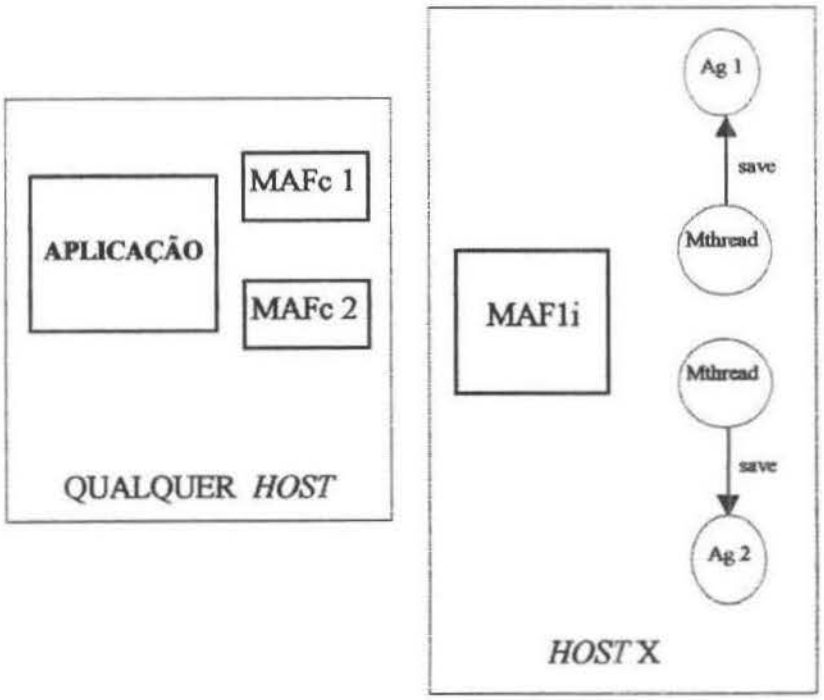

Figura 5.5 - Captura do estado de execução dos agentes.

Cada **Mthread** invocará no **MAF2i** o método *receive_agent*, passando entre outros parâmetros, o estado de execução dos agentes, o itinerário atualizado com a retirada de X e o nome do *host* onde os agentes foram criados (Figura 5.6).

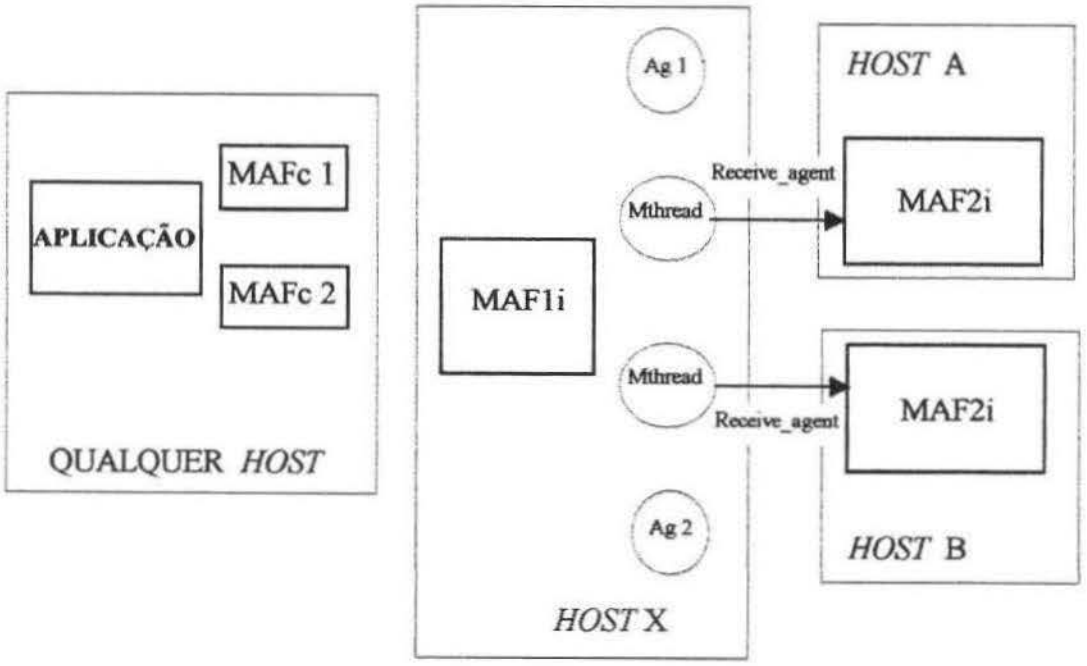

Figura 5.6 - Passagem dos parâmetros para a instanciação dos agentes em MAF2i.

A seguir, exatamente como **MAF1i**, **MAF2i** cria os **Mthreads** para os agentes (Figura 5.7).

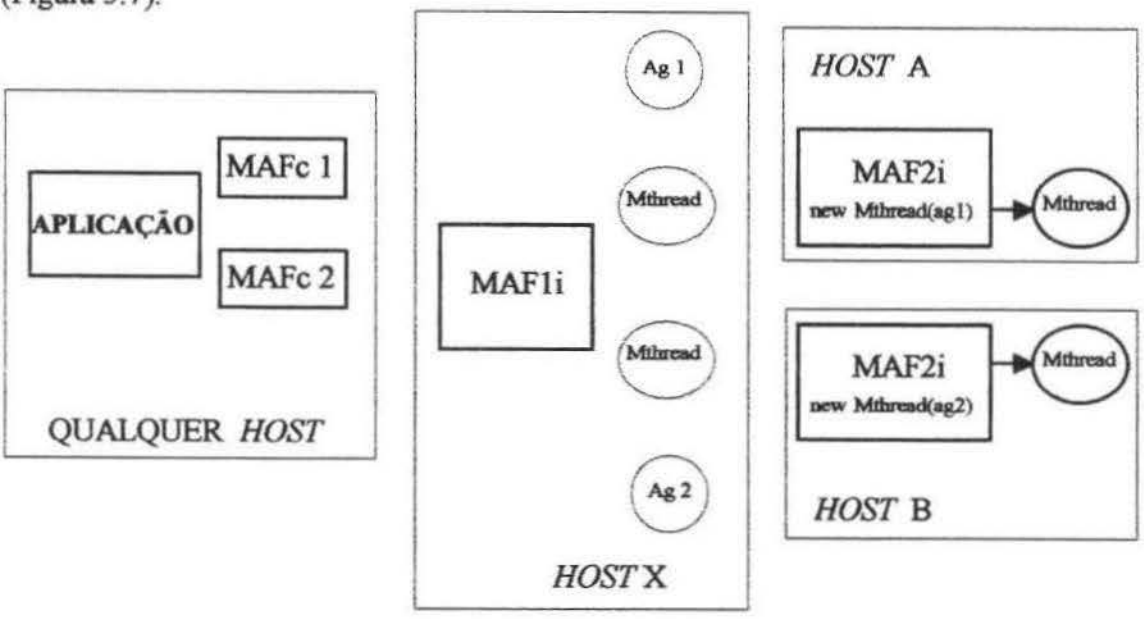

Figura 5.7 - Criação de Mthreads por MAF2i.

A diferença estará no servidor do agente que verificará se existe um estado a ser recuperado e instanciará o agente. A seguir, **MAF2i** invocará **MAF1i** no *host* inicial do agente e passará para este a referência atual do agente através do método *new_ref* (Figura 5.8).

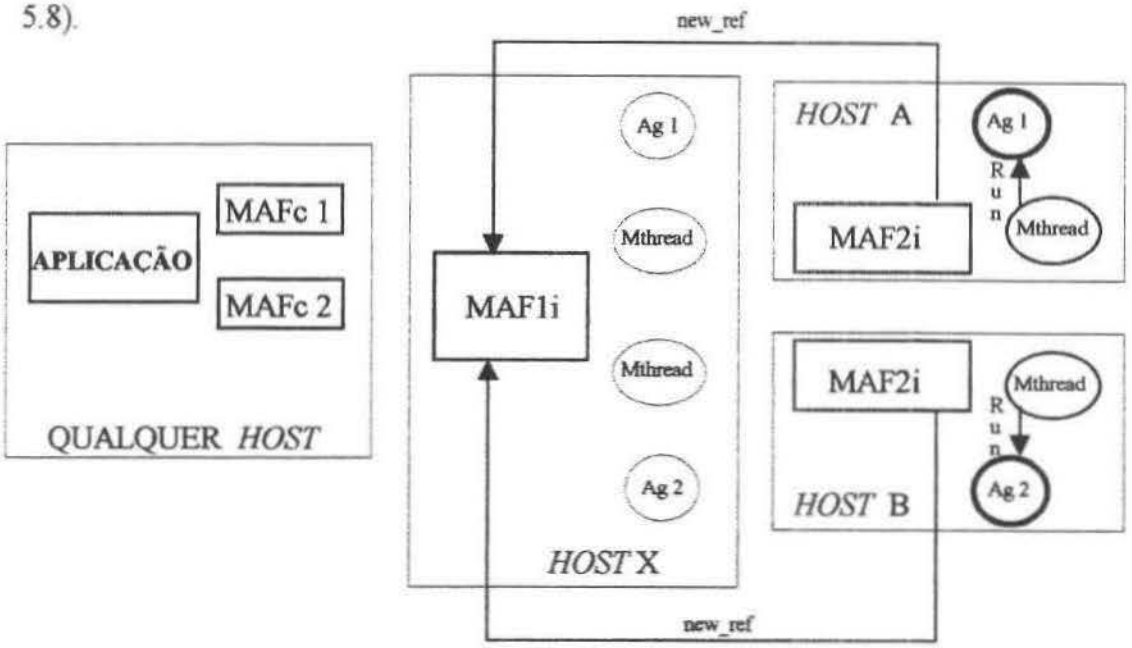

Figura 5.8 - Passagem da referência atualizada do agente para MAF1i.

Com o intuito de deixar claro os passos descritos acima, o cenário completo é mostrado na Figura 5.9.

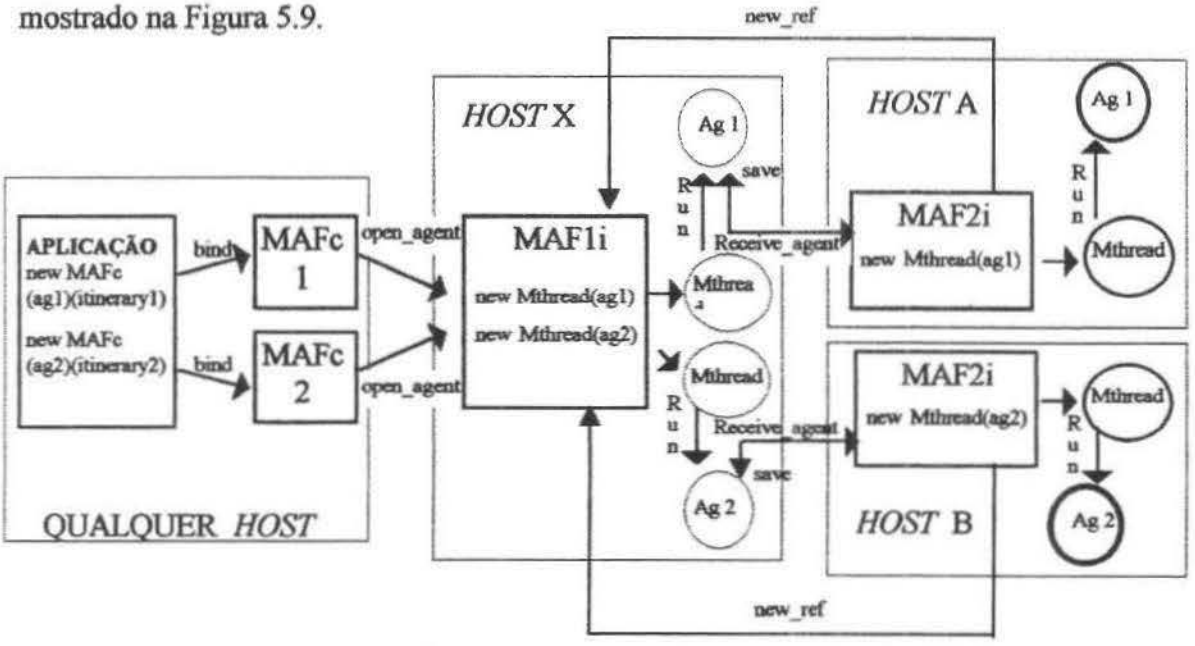

Figura 5.9 - Cenário completo do lançamento de dois agentes.

Conforme mostrado na Figura 5.9, 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 sua referência em **MAF1i**, a `table_ref` 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 de Java permitiu não somente a generalização das variáveis dos agentes, mas também do conteúdo das mensagens.

Outra contribuição do nosso suporte foi prover métodos para que um *Vector* fosse transparentemente transformado em uma *sequence* de *bytes* e vice-versa. Assim, o desenvolvedor de um agente não se preocupa com as características relacionadas a IDL e tão somente trabalha em Java.

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 (FEEC) e o do Instituto de Computação (IC). Os ORBs foram instalados nos seguintes domínios:

IC : `dcc.unicamp.br`

FEEC : `dca.fee.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.fee.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.

5.5 Situações de uso de agentes

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 5.10.

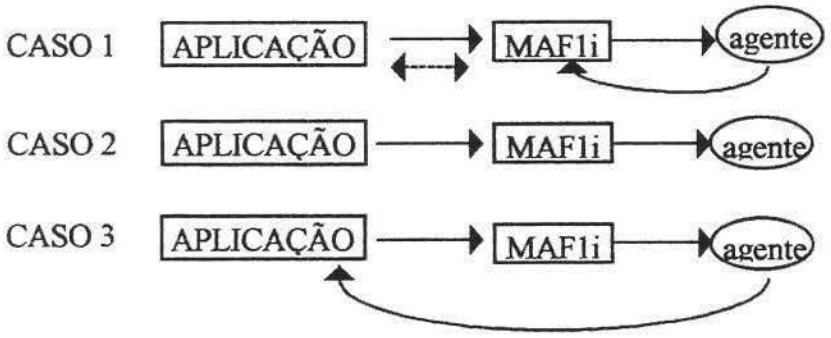

Figura 5.10 - Situações de uso de agentes.

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, é também atendido pelo sistema.

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* [Iona I, 1996]. Criamos então a classe abstrata **Callbackabs** que deverá ser especializada para cada tipo de aplicação cliente. 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 se quiser a invocação de métodos nos agentes concomitantemente com 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, este 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**.

5.6 Comunicação entre agentes

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. Normalmente, dois agentes que queiram se comunicar são oriundos de uma mesma aplicação e já conhecem a referência do **MAF1i** do outro agente. No caso de um agente necessitar comunicar-se com um agente de outra aplicação, a referência de objeto necessária deverá ser obtida de alguma forma, como por exemplo, através de uma invocação a outro objeto ou através de um *trader*.

A comunicação se dará como na Figura 5.11. Em um cenário semelhante ao usado anteriormente, o agente 1 quer se comunicar com o agente 2 e para isto busca a referência do mesmo através de ***get_ref*** (ag2).

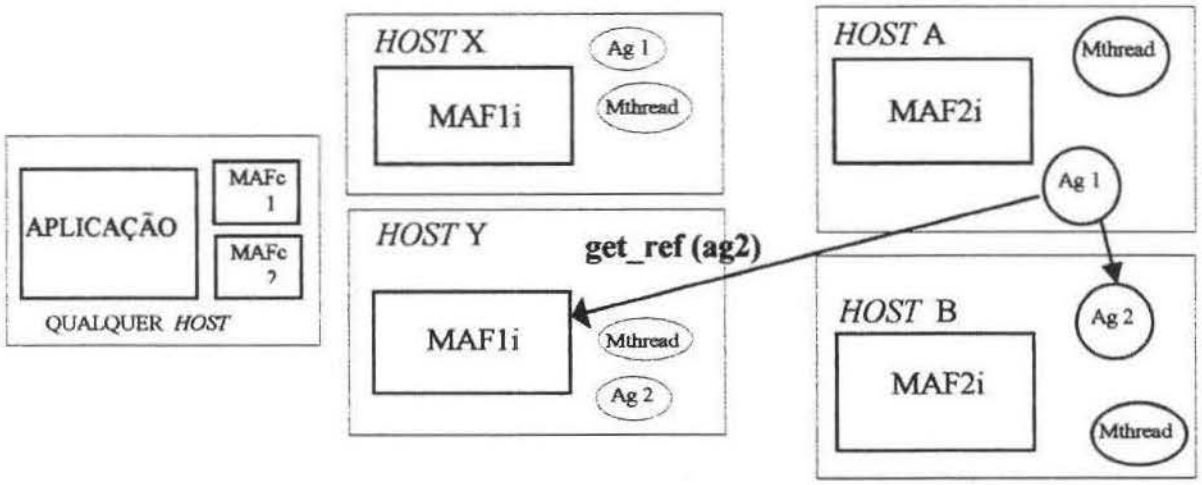

Figura 5.11 - Comunicação entre agentes.

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`.

Capítulo 6

Conclusão

A conclusão é apresentada dividida, tratando da validade do tema, da modelagem, da implementação e, por fim dos trabalhos futuros.

6.1 Importância do tema

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 suporte a agentes móveis para CORBA usando mecanismos do ORB para prover o meio de transporte para a migração dos agentes permite aliar-se 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.

6.2 Contribuições da Modelagem

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.

A principal contribuição da modelagem é a definição de um modelo para o suporte a mobilidade de agentes em um ORB e o suporte à comunicação dos agentes com outros agentes através de mecanismos padrões de CORBA [Vasconcellos et Madeira, 1998].

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.

Apresentamos também uma solução para o mapeamento de referências de Objetos Corba e de um mecanismo para a passagem destes Objetos CORBA por valor, usando estruturas definidas na própria especificação.

Por fim, nossa modelagem permite aos agentes, como qualquer outro Objeto CORBA, interagirem com outros agentes e Objetos CORBA estáticos, utilizando todas as facilidades providas pelo uso de um *broker*.

6.3 Contribuições da Implementação

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*.

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.

O MAF desenvolvido para a Multiware veio propiciar a pesquisa e o desenvolvimento de trabalhos por outros alunos conforme apresentado em [Ropelatto et al., 1998] [Ropelatto et Madeira, 1998] e [Schulze et Madeira,1999].

6.4 Trabalhos Futuros

Conforme citado no capítulo 4, deixamos para trabalhos futuros os aspectos relacionados com a segurança dos agentes móveis, das agências e de suas interações.

Como este trabalho foi desenvolvido utilizando-se um broker com mapeamento Java-IDL ainda não padronizado e, visto que esta padronização ocorreu após o término da implementação, sugere-se a futura migração do código do MAF para a OrbixWeb 3.

Por fim, o mapeamento de referências apresentado como solução para o controle de migração dos agentes pode e deve ser substituído por um controle utilizando-se o Serviço de Nomes da OrbixWeb visto que a vida do agente ficou dependente no qual este foi criado. Este Serviço somente foi disponibilizado após a conclusão deste trabalho.

Referências Bibliográficas

[Arnold et Gosling, 1996]
- Arnold,K. and Gosling,J. - The Java Programming Language. Sun Microsystems - 1996.

[Baker, 1997]
- Baker,S. – Bridging Boundaries: CORBA in Perspective - IEEE Internet Computing – setembro/outubro 1997 disponível em [URL 09].

[Benda, 1997]
- Benda,M. – Middleware: Any Client, Any Server - IEEE Internet Computing – julho/agosto 1997 – disponível em [URL 09].

[Berbers et al., 1996]
- Berbers,Y., De Decker,B. e Joosen,W. - Infrastructure for mobile agents – KULeuven – Department of Computer Science – disponível em [URL 17]/publications-E.html.

[Bernstein, 1996]
- Bernstein,P.A. – Middleware: A Model for Distributed System Services – Communications of the ACM – Vol. 39 N.2 – fevereiro 1996 – pp 86 – 98.

[Castelfranchi, 1996]
- Castelfranchi,C. - "To Be or Not to Be an "Agent""- Lecture Notes in Artificial Intelligence 1193 - agosto 1996 - pp 37-39.

[Chess et al., 1994]
- Chess,D., Harrison,C. e Kershenbaum,A. - Mobile Agents: Are they a good idea ? - IBM Research Report RC 19887- dezembro 1994 – disponível em [URL 05].

[Chess et al., 1995]
- Chess,D., Grosof,B., Harrison,C., Levine,D., Parris,C. e Tsudik,G. – Itinerants Agents for Mobile Computing - IBM Research Report RC 20010 - março 1995 – disponível em [URL 05]/rc20010.ps.

[Chess et al., 1997]
- Chess,D., Harrison,C. e Kershenbaum,A. - Mobile Agents: Are they a good idea ? update - Lecture Notes in Computer Science 1222 - abril 1997 - pp 46-47.

[Chung et al., 1997]
- Chung,P.E., Huang,Y., Yajnik,S., Liang,D., Shih,J.C., Wang,C. e Wang,Y. - DCOM and CORBA Side by Side, Step by Step and Layer by Layer - Bell Laboratories 1997 - disponível em [URL 14].

[Curtis, 1997] • Curtis,D. – Java, RMI and CORBA - OMG white paper – disponível em [URL 14].

[Ferreira, 1986] • Ferreira, Aurélio B. H. - Novo Dicionário Aurélio da Língua Portuguesa - 2ª edição - 1986 – Editora Nova Fronteira.

[Flanagan, 1997] • Flanagan, D.- Java in a Nutshell - 2nd Ed.- O'Reilly & Associates - maio 1997.

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

[Gage, 1997] • Gage,D. - OMG to Push CORBA Through ISO ? - Computer Reseller News - julho 1997 – disponível em [URL 18].

[Gray, 1995] • Gray,R. – Agent Tcl: Alpha Release 1.1 – Department of Computer Science - Dartmouth College - dezembro 1995.

[Gray et al.,1996] • 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 - disponível em [URL 08].

[Green et al., 1997] • Green,S., Hurst,L., Nangle,B., Cunningham,P., Somers,F. e Evans,R. - "Software Agents: A review" - Trinity College Dublin - maio 1997 – disponível em [URL 25].

[Harker,1995] • Harker,K. - TIAS: A Transportable Intelligent Agent System – Technical Report PCS-TR95-258 Department of Computer Science - Dartmouth College – 1995 – disponível em [URL 08].

[Iona I, 1996] • OrbixWeb Programming Guide - Iona Technologies Ltd.- novembro 1996 - disponível em [URL 02].

[Iona II, 1996] • OrbixWeb Reference Guide - Iona Technologies Ltd. - novembro 1996 - disponível em [URL 02].

[ISO, 1995a] • ISO/IEC 10746-1 ODP Reference Model Part 1 - Overview – 1995.

[ISO, 1995b] • ISO/IEC 10746-2 ODP Reference Model Part 2 – Foundations - 1995.

[ISO, 1995c] • ISO/IEC 10746-3 ODP Reference Model Part 3 – Architecture - 1995.

[Kiniry et Zimmerman, 1997] • Kiniry,J. e Zimmerman,D. - A Hands-on Look at Java Mobile Agents - IEEE Internet Computing – julho/agosto 1997 - [URL 09].

[Kotay et Kotz, 1994] • Kotay,K. e Kotz,D. - Transportable Agents - Department of Computer Science - Dartmouth College - novembro 1994 – disponível em [URL 16].

[Krause et al., 1997] • 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 – Proc. of 3rd International Symposium on Autonomous Decentralized Systems - 1997 - pp 93-102.

[Lingnau et al., 1995] • Lingnau,A., Drobnik,O. e Domel,P. - "An HTTP-based Infrastructure for Mobile Agents" - Fourth International WWW Conference – Boston - dezembro 1995.

[Lingnau et Drobnik,1995] • Lingnau,A. e Drobnik,O. - "An Infrastructure for Mobile Agents: Requirements and Architecture" - Fachbereich Informatik - Johann Wolfgang Goethe-University – Frankfurt – Germany – 1995 - disponível em [URL 20].

[Loyolla et al., 1994] • Loyolla,W., Madeira,E., Mendes,M., Cardozo,E., e Magalhães,M. – Multiware Platform: an Open Distributed Enviroment for Multimedia Cooperative Applications - IEEE COMPSAC – Taiwan - 1994.

[Mendes et Madeira, 1994] • Mendes,M. e Madeira,E.- Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware - 12° Simpósio Bras.de Redes de Computadores – 1994 – pp.93-109.

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

[Milojicic et al., 1998] • Milojicic,D., Breugst,M., Busse,I., Campbell,J., Covaci,S., Friedman,B., Kosaka,K., Lange,D., Ono,K., Oshima,M., Tham,C., Virdhagriswaran,S. e White,J. – MASIF The OMG Mobile Agent System Interoperability Facility – Proceedings of MA 98 - 1998.

[Milojicic, 1997]
- Milojicic,D. – Alternatives to Mobile Code - The Case for Mobile Agents – The Open Group Research Institute – fevereiro 1997 – disponível em [URL 06]/arpa.fr.ps.

[Milojicic et al., 1996]
- Milojicic,D., Bolinger,D., Zurko,M.E. e Mazer,M.S. – Mobile Objects and Agents - The Open Group Research Institute - novembro 1996 - [URL06]/moa.ps.

[Muller, 1996]
- Muller,N. - "Systems Management: The Emerging Role of Intelligent Agents"- Strategic Information Resources – disponível em [URL 24].

[Nwana et Wooldridge, 1997]
- Nwana,H. e Wooldridge,M. - "Software Agents Technologies" – Lecture Notes in Artificial Intelligence 1198 - 1997 - pp 59-78.

[OMG cf /95-11-03]
- OMG Doc cf/95-11-03 – Common Facilities RFP 3.

[OMG cf /96-08-01]
- OMG Doc cf/96-08-01 - Mobile Agent Facility Specification - Submission of IBM Co.

[OMG cf /96-08-02]
- OMG Doc cf/96-08-02 - Mobile Agent Facility Specification - Submission by Cristaliz Inc.

[OMG cf /97-06-04]
- Crystaliz, General Magic, GMD FOKUS, IBM Joint Submission - Mobile Agent Facility Specification.

[OMG formal/98-07-10]
- CORBA Facilities Architecture - Revision 4.0 – 1998.

[OMG formal/98-02-33]
- CORBA / IIOP 2.2 Specification - Full version – 1998 - disponível em [URL 01].

[OMG orbos/96-06-14]
- Objects-by-value Request For Proposal – 1996.

[OMG orbos/97-03-08]
- Java-to-IDL Request for Proposal – 1997.

[OMG orbos/97-05-25]
- CORBA Component Imperatives – maio 1997 - [URL 01].

[OMG orbos/97-10-05]
- Mobile Agent System Interoperability Facilities Specification Joint Submission by GMD Fokus, IBM co., Crystaliz, General Magic and The Open Group – 1997.

[OMG orbos/98-01-18]
- Objects by Value Joint Revised Submission – janeiro 1998.

[Orfali et al., 1996]
- Orfali,R., Harkey,D. e Edwards,J. - The Essential Distributed Objects Survival Guide - John Wiley & Sons- 1996.

[Orfali et al., 1997]
- Orfali,R., Harkey,D. e Edwards,J. - CORBA, Java and Object Web. The Web is in trouble. CORBA and Java are out to save it. - Byte Magazine – outubro 1997 disponível em [URL 10].

[Orfali et Harkey, 1997a]
- Orfali,R. and Harkey,D. – Client/Server Programming with Java and CORBA – 1st Edition - John Wiley & Sons – 1997.

[Orfali et Harkey, 1997b]
- Orfali,R. and Harkey,D. – CORBA, Java and Object Web – Software Development – abril 1997.

[Petrie, 1996]
- Petrie,C. - "What Is an Agent?" - Lecture Notes in Artificial Intelligence 1193 - agosto 1996 - pp 41-43.

[Petrie, 1997]
- Petrie,C. -"What's an agent...and what's so intelligent about it?" - IEEE Internet Computing - julho/agosto 1997 – disponível em [URL 09].

[Resnick, 1997]
- Resnick, R.I. - Intergalatic Distributed Objects - Dr.Dobb's Sourcebook magazine – janeiro/fevereiro 1997 - pp 35-39.

[Ropelatto et al., 1998]
- 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.

[Ropelatto et Madeira, 1998]
- Ropelatto,P. e Madeira,E. – Gerenciamento de Sistemas Distribuídos em Ambiente CORBA usando Agentes Móveis – II Workshop em Sistemas Distribuídos – II WoSiD – Curitiba – junho 1998 – pp.55-62.

[Rumbaugh et al., 1991]
- Rumbaugh,J., Blaha,M., Lorensen,W., Premerlani,W. and Eddy, F. - Object-Oriented Modeling and Design - Prentice-Hall, NJ. – 1991

[Rymer, 1996]
- Rymer,J.R. - The Muddle In The Middle - Byte magazine vol. 21 N.4 - abril 1996 - pp 67-70.

[Schulze et Madeira, 1997]
- Schulze,B. e Madeira,E. - Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture – Proceedings of Mobile Agents 97 – Lecture Notes in Computer Science 1219 – pp 74-85.

[Schulze et Madeira, 1999] • Schulze,B. e Madeira,E. – Migration Transparency in Agents Systems – IEEE Fourth International Symposium on Autonomous Decentralized Systems – ISADS'99 – Tóquio – Japão – março 1999 (aceito para publicação).

[Souza, 1998] • Souza,B. – JavaBeans – Componentes Java - Developers' Magazine – N.18 - fevereiro 1998 – pp 50-51.

[Straßer et al., 1996] • Straßer,M. , Baumann,J. e Hohl,F. – Mole: A Java Based Mobile Agent System - Institute for Parallel and Distributed Computer Systems – University of Stuttgart – outubro 1996 - disponível em [URL 19]/1996-straber-01.ps.

[Tanenbaum, 1996] • Tanenbaum,A. - Computer Networks – Third Ed.- Prentice-Hall, NJ. – 1996.

[Vasconcellos et Madeira, 1998] • Vasconcellos,F. e Madeira,E - Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA – 16° Simpósio Brasileiro de Redes de Computadores- Rio de Janeiro- maio 1998- pp501-520.

[Vinoski, 1997] • Vinoski,S. – CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments - IEEE Communications Magazine - Vol. 14 N.2 - fevereiro 1997.

[Vinoski, 1998] • Vinoski,S. – New Features for CORBA 3.0 – Communications of The ACM – Vol 41 N.10 – outubro 1998.

[Vogel et Duddy, 1997] • Vogel,A. e Duddy,K. Java Programming with CORBA - John Wiley & Sons – 1997.

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

[Yang et Duddy, 1996] • Yang,Z. e Duddy,K. - CORBA: A Platform for Distributed Object Computing - ACM Operating System Review Vol. 30 N.2 - 1996 - disponível em [URL 21].

URLs

[URL 01] http://www.omg.org
[URL 02] http://www.iona.com
[URL 03] http://www.ikv.de/download/index.html
[URL 04] http://www.omg.org/cgi-bin/membersearch.pl
[URL 05] http://www.research.ibm.com/massive/
[URL 06] http://www.osf.org/~dejan/papers/
[URL 07] http://www.objectspace.com/products/voyager
[URL 08] ftp://ftp.cs.dartmouth.edu/pub/CS-techreports
[URL 09] http://computer.org/internet/
[URL 10] http://www.byte.com/art/9710/sec6/art3.htm
[URL 11] http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html
[URL 12] http://www.tacoma.cs.uit.no/
[URL 13] http://www.trl.ibm.co.jp/aglets/
[URL 14] http://www.omg.org/library/whitepapers.html
[URL 15] http://www.genmagic.com/technology/mobile_agent.html
[URL 16] http://www.cs.dartmouth.edu/~agent/papers
[URL 17] http://www.cs.kuleuven.ac.be/cwis/research/hermix/index-english.html
[URL 18] http://www.crn.com/print-archive/19970728/747news027.asp
[URL 19] http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html
[URL 20] http://www.tm.informatik.uni-frankfurt.de/agents/
[URL 21] http://www.infosys.tuwien.ac.at/Research/Corba/intro.html
[URL 22] http://www.cnri.reston.va.us/home/koe/bib/mobile-abs.bib.html
[URL 23] http://www.cl.cam.ac.uk/users/rwab1/ag-people.html
[URL 24] http://www.ddx.com/agents.shtml
[URL 25] http://www.cs.tcd.ie

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);
};
```

Anexo B : Agentserver

Classe base para a confecção do servidor do agente.

Cada agente deverá ter um servidor registrado (putit) indicando uma classe como esta.

Substituir as partes do código entre <...> por nomes especificos

```
package <agentpackage>;
import MAF.*;
import IE.Iona.Orbix2._CORBA;
import IE.Iona.Orbix2.CORBA.SystemException;
import java.io.IOException;
import java.io.*;

public class <Agentserver> {

        public static void main (String args[]) {

                String s;
                _AgenteRef cTie;
                <Agentimplementation> agent;
                String server;
                String marker;

                try {
                        server=_CORBA.Orbix.myImplementationName ();
                }
                catch (SystemException se) {
                        System.out.println ("get server failed");
                        System.out.println (se.toString());
                        return;
                }

                try {
                        marker=_CORBA.Orbix.myMarkerName ();
                }
                catch (SystemException se) {
                        System.out.println ("get marker failed");
                        System.out.println (se.toString());
                        return;
                }
```

```
            s = server + marker;
            File fi = new File("/tmp", s);

            if (fi.exists()){
                try{
                    FileInputStream f = new FileInputStream(fi);
                    ObjectInputStream ob = new ObjectInputStream(f);
                    agent = (<Agentimplementation>) ob.readObject();
                }
                catch (Exception e){
                    System.out.println ("erro ao recuperar o arquivo");
                    e.printStackTrace();
                    return;
                }
            }
            else agent = new <Agentimplementation>();

            try {
                cTie= new _tie_Agente (agent, marker);
            }
            catch (SystemException se) {
                System.out.println ("TIE failed");
                System.out.println (se.toString());
                return;
            }

            try {
                _CORBA.Orbix.impl_is_ready ();
            }

/* O SERVER DEVE TER SIDO REGISTRADO NO REPOSITÓRIO DE
IMPLEMENTAÇÕES */

            catch (SystemException se) {
                System.out.println ("impl_is_ready failed");
                System.out.println (se.toString());
                return;
            }
        }
}
```