Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Departamento de Engenharia de Computação e Automação Industrial

"Uma abordagem para o transporte de agentes móveis em ambientes abertos, heterogêneos e distribuídos"

Carlos Raúl Arias Méndez

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação, da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Doutor em Engenharia Elétrica e Computação.

Banca examinadora:

Prof. Dr. Manuel de Jesus Mendes (orientador)
Prof. Dr. Juan Manuel Adán Coello
Prof. Dr. Edmundo Roberto Mauro Madeira
Prof. Dr. Mauricio Ferreira Magalhães
Prof. Dr. Mário Jino

Campinas, SP, agosto de 2000

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Ar41a

Arias Méndez, Carlos Raúl
Uma abordagem para o transporte de agentes móveis em ambientes abertos, heterogêneos e distribuidos / Carlos Raúl Arias Méndez.--Campinas, SP: [s.n.], 2000.

Orientador: Manuel de Jesus Mendes.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Arquitetura de redes de computador. 2. Programação orientada a objetos (Computação). 3. Sistemas operacionais distribuídos (Computadores). I. Mendes, Manuel de Jesus. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

O desenvolvimento de aplicações em redes como a Internet, está exigindo novos requerimentos que as atuais tecnologias de software não estão facilitando o desenvolvimento destas aplicações em forma eficiente e flexível, emergindo o paradigma do agente móvel como uma alternativa para abordar estes novos requerimentos.

O presente trabalho de tese de doutorado visa fazer uma análise destes requerimentos, concentrando-se nas questões associadas ao transporte de agentes através de um meio de comunicação aberto, heterogêneo e de alto tráfego, como a Internet.

Como produto desta análise propõe-se um serviço para o transporte de agentes, apresentando suas funcionalidades e arquitetura, podendo ser implementado em um ambiente aberto (baseado em CORBA), heterogêneo e distribuído.

Discute-se sua implementação, fazendo-se uma análise dos resultados obtidos dos testes de desempenho realizados na Internet e em uma intranet, quando se realiza uma transferência de arquivos (representando os agentes) e se simula uma aplicação Cliente/Servidor, comparando ambos os modelos a partir destes resultados.

Abstract

The development of applications in global networks as the Internet, is demanding new requirements that the current software technologies are not providing to build these applications in an efficient and flexible way. But, as an alternative to this new requirements is emerging the mobile agent paradigm.

The present doctoral thesis work seeks to do an analysis of these requirements, concentrating on the subjects associated to the agent transport through an open, heterogeneous, and high traffic communication network, as the Internet is.

After doing the analysis, we propose an agent transport service based on CORBA technology, describing its architecture and functionalities, that could be implemented in an open, heterogeneous and distributed environment.

We discuss its implementation, being done an analysis of the test results that were accomplished in the Internet and in a intranet, when a file transfer is taking place (representing the agents) and when a Client/Server application was simulated, comparing the results from both models.

Palavras chaves:

Agentes móveis, CORBA, transporte de agentes, sistemas abertos, orientação a objetos.

Agradecimentos

Desejo manifestar meus mais sinceros agradecimentos a todas as pessoas que de uma ou outra forma ajudaram nesta jornada e aquelas instituições que, com o apoio financeiro, permitiram-me financiar este trabalho. Agradeço em forma especial:

* A meus amigos Diego e José.
* A meu professor orientador Dr. Manuel de Jesus Mendes pelo seu incentivo e apoio durante a orientação deste trabalho, especialmente pela sua infinita paciência para corrigir os erros de redação que cometi devido a que o idioma português não é a minha língua materna.
* Ao CNPq
* A Universidad de Magallanes

Finalmente, agradeço a minha esposa Rosa e filhos Alejandra, Rodrigo e Andrea pela paciência e apoio durante o desenvolvimento deste trabalho, para eles dedico esta tese.

Índice

Parte I: Aspectos teóricos e projetos relacionados com a mobilidade

1 Introdução

O desenvolvimento de tecnologias nos meios de comunicação e das redes de computadores está permitindo a integração dos diversos meios de informação: voz, áudio, vídeo e dados, além de permitir uma maior mobilidade nas comunicações. Isto desemboca no fornecimento de serviços mais sofisticados, resultando em sistemas cada vez mais complexos. Por outro lado, os usuários precisam que estes desenvolvimentos e mudanças de tecnologias não atrapalhem suas atividades nem encareçam seus orçamentos. Também, o crescimento destas redes, tanto as globais como a Internet [Etzioni95] e as privadas como a intranet, fazem que o tráfego dos dados esteja em constante crescimento, comprometendo o desempenho da mesma e desmotivando os usuários no uso destes meios.

Tudo isso justifica a pesquisa de novos modelos de programação para tentar aliviar estes problemas [Bagrodia95] [Baumann97] [CACM94] [Chang96] [Chess95] [Franklin96] [IAG97] [Mendes96]. Nessa direção os agentes móveis estão emergindo como um paradigma interessante, porque sua autonomia permite "injetá-los" na rede deixando que eles realizem o trabalho do usuário. Desta forma, os agentes podem viajar de um lugar para outro, baseados em um itinerário [Strasser98], onde localmente realizam a interação com os servidores, evitando assim um maior tráfego nas redes globais.

Atualmente existem inúmeros projetos de pesquisa em andamento além de alguns produtos comerciais [Kiniry97], baseados em linguagens de programação tradicionais e em novas linguagens com características de mobilidade incorporadas. Também existem esforços para permitir a interoperabilidade dos produtos de fabricantes diferentes; desta forma os usuários poderão tirar um maior proveito dos diversos recursos instalados na rede, para não ficar circunscritos só àqueles para os quais o produto comercial foi desenvolvido.

O paradigma do agente móvel está sendo desenvolvido principalmente utilizando linguagens que possuem características de mobilidade, como é o caso das linguagens Java, Telescript, Obliq, Agent Tcl, Tycoon, Facile. Estas linguagens apresentam suas próprias soluções para as questões que a mobilidade do código acarreta. É possível descrever um cenário onde existam aplicações de agentes móveis trabalhando na rede, baseadas em linguagens diferentes e utilizando a própria infra-estrutura oferecida por cada uma das linguagens. Esta situação cria redundância no uso dos recursos da rede, porque estas infra-estruturas vão ter, para cada tipo de linguagem, meios de transporte, registro de nomes de agentes, segurança, mecanismos de execução, etc., os quais poderiam se evitar ao considerar uma infra-estrutura única para os diferentes tipos de linguagens.

Por outro lado, este paradigma está sendo considerado pelos pesquisadores porque vislumbra vantagens [Bieszczad98a] [Bic97] [Nwana96] [Kotz99]. Mas estas vantagens só

poderão ser avaliadas se forem desenvolvidas plataformas para a realização deste modelo. Entre as vantagens que podem ser consideradas no paradigma estariam as seguintes:

- Permite-se que o agente execute independentemente de quem o envia (o cliente), operando de forma assíncrona.
- Como o agente atua pelo cliente, este último pode desligar-se da rede.
- O agente pode visitar vários nós, em vez de um só (por ex. um *applet*, após chegar no servidor, não poderá continuar indo para um outro).
- Permite-se reduzir o tráfego na rede, ao colocar-se a lógica da aplicação mais perto do servidor, para interagir com ele.
- Permite-se que os agentes possam se ajustar às capacidades do servidor, desempenho e condições de falha no hospedeiroou na rede.

Mas os agentes criam novos desafios que devem ser resolvidos, como por exemplo:

- Problema de Segurança: Pelo fato de um agente "visitar" sistemas computacionais, estes sistemas ficarão mais vulneráveis aos ataques de entidades maliciosas. Por outro lado, os agentes estarão expostos aos possíveis ataques dos sistemas que os acolhem.
- Identificação: É difícil identificar-se univocamente o agente e seguir sua trajetória na rede.
- Infraestrutura:Torna-se necessário construir uma infra-estrutura para o uso e controle dos agentes, por exemplo, para a negociação do movimento, interfaces para ativar, suspender ou eliminar um agente, etc.
- Reengenharia das Aplicações: As aplicações deverão se adaptar para o uso dos agentes, como um novo paradigma de programação.
- Padronização: É conveniente a definição de uma infra-estrutura padronizada para o suporte de agentes.

Dentro das aplicações [Harrison95] [IAG97] que podem ser consideradas no uso dos agentes móveis podem-se mencionar:

- Busca da informação distribuída [Dale97]
- Comércio eletrônico [Villinger96] [Gollmann98]
- Agentes inteligentes
- Assistentes pessoais
- Computação móvel [Hurst97]

- Gerenciamento de redes [Susilo98] [Bic99]
- Computação de propósito geral
- Atividades de coordenação
- Workflow
- Telecomunicações [Breugst98]

Esta tese tem como objetivo analisar as questões envolvidas no transporte de agentes, fazer um estudo das funcionalidades que uma plataforma deve possuir para suportar a mobilidade dos agentes, e propor um serviço de transporte de agentes que satisfaça os requerimentos. O serviço desenvolvido formará parte de um conjunto de serviços que visa construir uma plataforma geral para o suporte de mobilidade de agentes. Após desenvolvimento da proposta, apresentam-se os resultados dos testes para medir o desempenho dos protocolos de transporte, e demonstra-se sob quais condições o paradigma de agente móvel apresenta um melhor desempenho do que o modelo clássico de C/S.

Nesta tese, considera-se o desenvolvimento de uma plataforma aberta e distribuída, com serviços de suporte para a mobilidade e interoperabilidade de agentes móveis de diversos fabricantes. Para isso, apresenta-se um serviço que permite o transporte de agentes móveis, baseado na arquitetura aberta CORBA da OMG, a qual suporta o desenvolvimento de aplicações distribuídas orientadas a objetos. O serviço desenvolvido poderá ser utilizado por qualquer aplicação baseada em agentes móveis desde que a linguagem de programação tenha o mapeamento com IDL definido (C++, C, Ada, Java, Smalltalk).

Quando se fala em uma plataforma aberta, quer dizer-se que sua arquitetura é pública para permitir que agentes de fabricantes diferentes possam se comunicar. Desta forma, a plataforma pode crescer, sem restrições, oferecendo as vantagens dos esquemas abertos, qualidade e concorrência nos custos e tendo os usuários maiores possibilidades para escolha, de acordo com seus orçamentos e objetivos.

Quando se fala de uma plataforma distribuída, está-se pensando em uma arquitetura que permita flexibilidade na implementação, no sentido que os serviços de suporte para a mobilidade de agentes móveis possam ser indistintamente implementados em um computador pessoal, estação de trabalho, em uma rede local ou em qualquer outra estrutura de computadores. A arquitetura CORBA, através dos diversos ORBs que já existem no mercado, permite implementar estes esquemas, de forma natural.

Durante o desenvolvimento deste trabalho foram analisados alguns projetos típicos na área, fazendo ênfases no transporte dos agentes móveis. Também, foram consideradas ferramentas de software e linguagens de programação existentes no mercado, para analisar sua aplicabilidade na implementação do serviço proposto.

A apresentação deste trabalho foi estruturada em três partes:

- Parte I: Aspectos teóricos e projetos relacionados com a mobilidade.
- Parte II: Desenvolvimento de um Serviço de Transporte de Agentes.
- Parte III: Discussão de resultados e comentários finais.

A parte I abrange os quatro capítulos seguintes:

- Capítulo 1: Introdução, em que se faz a apresentação do contexto e dos objetivos do trabalho.
- Capítulo2: Código móvel e agentes, em que se apresentam os conceitos e questões associados à área de agentes móveis e se define o escopo do trabalho.
- Capítulo 3: Plataformas para o suporte da mobilidade. Descrevem-se ferramentas de software que estão relacionadas às questões de mobilidade e analisa-se sua aplicabilidade à implementação de plataformas para o suporte de agentes móveis.
- Capítulo 4: Aspectos de transporte em projetos de agentes móveis. Descrevem-se e analisam-se projetos na área de agentes móveis, discutindo-se estes projetos sob as principais questões relacionadas com o transporte de agentes e finalizando com a justificação para a proposta de um serviço de transporte de agentes.

A parte II abrange os três capítulos seguintes:

- Capítulo 5: Funcionalidades do STA. Estabelece-se os requerimentos e as funcionalidades de um STA, finalizando com a proposta de uma arquitetura geral para este serviço.
- Capítulo 6: Operação e especificação do STA. Descreve-se a operação deste serviço e sua relação com outros serviços. Finalmente explicam-se os critérios utilizados para escolher as interfaces e sua especificação em IDL.
- Capítulo 7: Aspectos de implementação de um STA. Desenvolve-se um exemplo parcial deste serviço com o objetivo de fornecer uma visão prática dos aspectos de implementação.

Finalmente, a parte III abrange os dois capítulos seguintes:

- Capítulo 8: Comparação dos modelos do Agente Móvel e do Cliente /Servidor. Apresentam-se e discutem-se os resultados obtidos nos testes baseados no transporte de arquivos e na simulação de uma aplicação Cliente/Servidor, através da Internet e de uma intranet. Baseado nos resultados obtidos, comparam-se ambos os modelos.
- Capítulo 9: Conclusões finais e trabalho futuro. Apresentam-se as conclusões finais que se obtiveram e também apresenta outros aspectos que seriam interessantes de abordar, visualizados no término deste trabalho.

2 Código móvel e agentes

Neste capítulo explicam-se diversos conceitos, termos e mecanismos relacionados com agentes móveis. Alem disso, identificam-se os problemas básicos da implementação de aplicações baseadas neste paradigma.

Primeiramente, é importante salientar que existe uma área maior que contém os agentes móveis, designada na literatura por **código móvel**. Este termo diz respeito à característica que se pode incorporar ao programa de uma aplicação, para que ele, ou parte dele, se possa movimentar através de diferentes ambientes de execução. Estes ambientes de execução podem residir em um ou mais hospedeiros interligados em uma rede.

Ao falar-se de movimento de código pressupõe-se que, de alguma forma, o código estará presente na máquina destino, além de, opcionalmente, seu estado e provavelmente outros dados adicionais. Mas, para poder efetuar a mobilidade, é necessário que existam os mecanismos que permitam suportar esta característica.

A mobilidade do código vem sendo tratada na literatura já há cerca de 20 anos, e até agora pode-se dizer que está sendo estudada a partir de três abordagens diferentes:

- **Ao nível do sistema operacional:** Inicialmente começou-se a considerar a mobilidade do código quando se pesquisava a migração de processos [Milojicic00] [Liao96] [Johansen95] entre diferentes máquinas, no contexto de sistemas tolerantes a falhas e de distribuição de carga [Rubin96]. A migração destes processos significa que a execução de um processo é interrompida, seu estado de execução salvo (*checkpointing*), isto é, o conteúdo do contador do programa, do *stack* e dos registros internos da CPU é guardado e transportado junto com o código, para outra máquina, na qual o processo é reiniciado a partir do estado salvaguardado.

  Neste tipo de aplicações o ambiente de execução é homogêneo e aplicado nas redes locais. O controle acontece dentro do sistema operacional, que decide para onde e quando fazer a migração. Entre outros projetos nesta área pode-se citar Tacoma [Tacoma99] [Jacobsen99] [Johansen98].

- **Ao nível da linguagem de programação:** Posteriormente, considerou-se que essa mobilidade do código deveria estar disponível para os programadores das aplicações em rede, através das linguagens de programação, de forma a poder-se experimentar novas técnicas de programação para ambientes de redes cada vez mais dinâmicos. Neste nível, o escopo do ambiente de trabalho aumenta, considerando-se agora ambientes heterogêneos, tanto em hardware como software. Normalmente, a iniciativa da decisão de envio do código para outro computador, é deixada para a aplicação. Para contornar o problema da

portabilidade do código, desenvolveram-se linguagens do tipo *script* ou interpretadas, as quais permitem levar consigo seu próprio *stack* de comandos e o endereço do seguinte comando a executar, além dos valores das variáveis da aplicação móvel. Como exemplo, têm-se linguagens que possuem primitivas de mobilidade [Cugola96] dentro de seu repertório de instruções e que incorporam à linguagem novos comandos, modificando o compilador original. É o caso da linguagem Facile [Knabe95] que foi estendida para suportar a mobilidade de código, além de outras criadas especificamente para aplicações com características de mobilidade, como Telescript, desenvolvida pela General Magic Inc., [White98]. Também se desenvolveram linguagens semi-interpretadas, as quais são compiladas para um código intermediário e transportável para sistemas diferentes. O interpretador do código intermediário é desenvolvido para cada sistema alvo (máquinas virtuais), oferecendo-se assim alta portabilidade. Contudo, estes sistemas não incorporam ainda mecanismos de manipulação do estado de execução de suas máquinas virtuais que interpretam o código. Como exemplos têm-se a linguagem Java [Java96] e Oberon [Oberon99]. OBERON foi desenvolvida a partir da linguagem Pascal com a modificação do compilador para geração de código intermediário.

- **Ao nível da camada intermediária (*middleware*):** Para que a linguagem de programação possa ter características de mobilidade na sua construção, é necessário que ela seja desenvolvida levando em consideração o sistema operacional. Como alternativa propôs-se a possibilidade de abordar a mobilidade do código desenvolvendo uma camada intermediária entre a aplicação e o sistema operacional, a qual forneça a funcionalidade (APIs ou bibliotecas de funções) para que a aplicação possa fazer uso da mobilidade. Assim, ficam separadas as questões da mobilidade das questões próprias da linguagem. O projetista da linguagem não deve se preocupar com questões relacionadas com a mobilidade. Nesta área há uma grande quantidade de projetos que estão sendo ou foram desenvolvidos (Aglets, MOA, MOLE, Grasshopper e outros).

2.1 Classificação dos mecanismos de código móvel

Existem na literatura algumas classificações que permitem esclarecer as diversas opções que a mobilidade do código oferece. Uma das mais completas é a estabelecida pelos autores Fuggetta et al [Fuggetta98], que não necessariamente significa ser a mais adequada, mas que apresenta uma visão rápida e sucinta desta área ainda em evolução. Neste trabalho apresenta-se uma outra classificação dos mecanismos de código móvel, considerando a iniciativa da transferência e os mecanismos de mobilidade.

Iniciativa na transferência: Considerando quem toma a iniciativa para que o código se movimente da fonte para o destino, existem duas possibilidades:

- ***Fetch***: O destino solicita para a origem enviar o código, por exemplo, os *applets*, os quais são carregados pelo *browser* a partir do servidor da aplicação. Outra situação apresenta-se quando um agente migra para outro local, e informa o destino para enviar seu código a partir de um endereço determinado;
- ***Forward***: A fonte conhece o endereço do destino e procede ao envio do código sob seu controle. Normalmente esta situação ocorre na maioria das aplicações que utilizam código móvel.

Mecanismos de mobilidade: Como já foi dito, não somente o código pode ser enviado ao destino, como também o estado da entidade que está sendo transferida. Além disso, o código poderá ser ou não ser transferido fisicamente de um lugar para outro, gerando diferentes situações, nas quais se combinam os casos anteriormente mencionados. Desta forma podem ser considerados os seguintes **mecanismos de mobilidade**:

- **Cópia**: No destino gera-se uma cópia do código original, com uma identificação diferente e sem considerar o estado de execução.
- **Movimento**: Este tipo de mobilidade faz que o código que está na origem seja instalado no destino, mantendo a mesma identificação e sem considerar seu estado de execução. Após o movimento, o código origem é apagado.
- **Migração**: Este caso é similar ao anterior. O estado atual do código também é transferido para o destino, sendo considerado na iniciação do programa no destino. Após o movimento, o código é deletado na origem, e passará a ser executado no destino mantendo sua identificação.

Para ilustrar as definições anteriores, considere-se uma aplicação que realize o mesmo trabalho em nodos distintos (por exemplo, atualização de um arquivo). Para tal envia-se uma cópia do código simultaneamente e de forma paralela para os diferentes nodos, ou, numa outra alternativa, para uma lista de endereços, referentes aos nodos que serão visitados de forma seqüencial (ou consoante prioridades preestabelecidas).

No primeiro caso, está se falando de enviar uma **cópia** do código para cada destino e no segundo caso, está se falando de **movimentar** o código entre os diferentes lugares que devem ser visitados.

Se a aplicação precisar de armazenar dados em cada um dos locais que visita para realizar algum tipo de processamento (por exemplo, coletar dados para achar o valor maior), fala-se de **migração** (no exemplo é necessário levar o "estado" da variável que contém o maior valor até então encontrado).

É importante salientar que, às vezes, mover o código não significa necessariamente movê-lo fisicamente. Por exemplo, quando considerar uma aplicação de distribuição de carga entre computadores clientes em uma rede local, onde todos eles estão ligados a um mesmo sistema de arquivos, o código que se quer movimentar estará sempre presente no disco do servidor. Portanto, não vai ser necessário mover o arquivo fisicamente de lugar, bastando carregá-lo, a partir do arquivo, no computador para o qual foi movimentado e onde será executado.

2.2 Conceitos de estado e de *marshalling*

Nesta seção discutem-se dois conceitos associados à mobilidade do código: estado e *marshalling*. O primeiro apresenta diferentes definições na literatura, dependendo do contexto. As diferentes explicações encontradas na literatura sobre o segundo conceito são pouco claras ou pouco explícitas.

2.2.1 Conceito de estado

Dependendo do nível em que está acontecendo a mobilidade, pode-se falar da mobilidade dos processos, do ponto de vista do sistema operacional, ou da mobilidade do código ou entidade (agente, objeto, função) na aplicação.

Em ambos os casos, o conceito de estado tem conotações diferentes. No contexto do sistema operacional, o estado refere-se a um conjunto de valores que descrevem a situação atual de execução de um processo:

- O valor que contém o registro contador de programa, relativo ao endereço da seguinte instrução que o processo ia executar ao acontecer o *checkpointing*;
- Os valores dos registros internos da CPU, incluindo-se o conteúdo do *stack pointer*;
- Os valores das variáveis do processo armazenadas no *stack*.

Neste nível, a transferência dos processos com estado é totalmente transparente para a aplicação, exigindo-se portabilidade ao nível de código binário entre os diferentes sistemas envolvidos na transferência, o que atualmente só acontece em sistemas homogêneos.

No contexto da aplicação, o estado pressupõe transferir os valores das variáveis da entidade de código que está-se movimentando, considerando-se o grau de atomicidade manipulado (comando, objeto, agente). Por exemplo, deve-se transferir a informação necessária para determinar no destino o comando seguinte a executar. Esta situação não se refere a instruções em código binário mas sim a comandos no código fonte. Por esta razão, a tendência é utilizar linguagens interpretáveis ou *scripts*. No caso do exemplo anterior, deveria escrever-se um *script* que ofereça facilidades para capturar a informação relativa à sentença seguinte a ser executada. Desta forma o controle da execução do programa fica

independente da máquina alvo. Outra alternativa opera com linguagens baseadas em máquinas virtuais, como é o caso de Java, de forma a assegurar a portabilidade. Mas a questão de conseguir salvar o estado de execução destas máquinas virtuais ainda não está resolvida, porque não é possível ter acesso ao contador de programa da máquina virtual. Só é possível salvar o estado das variáveis de forma automática, como acontece com a serialização dos objetos em Java, onde os valores dos atributos dos objetos são armazenados automaticamente quando a serialização é invocada.

No caso de um agente, este problema ainda é mais complexo. Primeiro, porque ele pode ser uma entidade de software, baseada em funções ou num conjunto de objetos. Em segundo lugar, o estado não necessariamente consiste somente dos valores dos atributos dos objetos ou das variáveis das funções que o compõem, mas também das relações que existam entre os objetos ativos ou da relação entre as funções ativas, no momento da migração.

Esta questão pode ser resolvida, de forma geral, no contexto da aplicação, construindo-se o programa baseado em máquina de transição de estados (como é o caso de protocolos). Desta forma, o programa agirá de acordo com os dados de entrada que receba. Contudo esta é uma solução na qual o programador é o responsável pela manipulação do estado da aplicação que está se movimentando. Além disso, obriga a utilizar uma estrutura baseada em uma máquina de estados que não sempre será adequada para a codificação de aplicações (por exemplo, quando o número de estados for muito grande).

O conceito de estado de um agente não se considera hoje totalmente esclarecido. À medida que sejam desenvolvidas aplicações baseadas em agentes, ganhar-se-á experiência para determinar os requerimentos gerais envolvidos no conceito de estado (de um agente) e para desenvolver metodologias para capturar o estado de forma transparente para o programador da aplicação.

2.2.2 Conceito do *Marshalling*

Como é sabido, a linguagem de programação oferece o uso de tipos de dados básicos ou primitivas, para que o programador as utilize na definição de seus próprios dados. Estes primitivas são representadas por uma certa quantidade de bytes. Por exemplo, em C++ os dados do tipo inteiro podem ser representados por 4 ou 8 *bytes* para *int* (dependendo da máquina) e 8 bytes para *long*; os dados do tipo real podem ser representados por 4 *bytes* para precisão simples e 8 *bytes* para precisão dupla; os dados do tipo caractere podem ser representados por 1 *byte*, etc. Ou seja, dentro de uma linguagem, o tamanho para um mesmo tipo de dados pode ser diferente, dependendo do tipo de máquina (por exemplo, PCs e máquinas de grande porte)

Quando os dados são processados e criados na mesma máquina, não existe problema nenhum na consistência dos mesmos. Mas quando estes dados devem ser processados em

máquinas diferentes (com CPU e/ou sistema operacional diferentes), podem acontecer problemas como os que se explicam na continuação.

Uma primitiva é representada por uma quantidade definida de bytes e, dependendo da CPU e/ou sistema operacional, estes bytes, quando são carregados em memória, podem ser ordenados de uma das seguintes formas:

- ***Big-endian*:** O *byte* mais significativo ocupa o primeiro lugar da memória.
- ***Little-endian*:** O byte menos significativo ocupa o primeiro lugar da memória.

Para melhor esclarecimento apresenta-se uma tabela de bytes ordenados para diferentes combinações de CPUs e sistemas operacionais.

CPU	**S.O.**	**Ordem**
Alpha	Todos	Little-endian
HP-PA	NT	Little-endian
HP-PA	UNIX	Big-endian
Intel X86	Todos	Little-endian
Motorola 680X0	Todos	Big-endian
MIPS	NT	Little-endian
MIPS	UNIX	Big-endian
PowerPC	NT	Little-endian
PowerPC	não-NT	Big-endian
RS/6000	UNIX	Big-endian
SPARC	UNIX	Big-endian

A seguir apresentam-se as representações de um dado de 4 bytes:

```
Bits: [ 7  6  5  4  3  2  1  0 ] Byte 0 < endereço X
Bits: [ 15 14 13 12 11 10 9  8 ] Byte 1 < endereço X+1
Bits: [ 23 22 21 20 19 18 17 16 ] Byte 2 < endereço X+2
Bits: [ 31 30 29 28 27 26 25 24 ] Byte 3 < endereço X+3
```

Ordenamento *Little-endian* para dado de 4-bytes.

Bits:	[31	30	29	28	27	26	25	24]	Byte 0	< endereço X
Bits:	[23	22	21	20	19	18	17	16]	Byte 1	< endereço X+1
Bits:	[15	14	13	12	11	10	9	8]	Byte 2	< endereço X+2
Bits:	[7	6	5	4	3	2	1	0]	Byte 3	< endereço X+3

Ordenamento *Big-endian* para dado de 4-bytes.

Quando um programa acessa um dado da memória, vai fazê-lo pegando o byte que está na primeira posição. Portanto, ao carregar na memória o valor de um dado que foi criado em um sistema *big-endian* e que posteriormente vai ser processado em um sistema *little-endian*, exigirá a inversão da seqüência dos bytes, para que a semântica do dado não perca consistência para a aplicação.

Logo, quando os dados de uma aplicação vão ser serializados em um *stream* e transportados para outra máquina, a questão do ordenamento dos bytes deverá ser considerada. Existem várias formas para abordar esta questão. Numa delas escrevem-se os dados em ASCII, indicando: seu tipo, tamanho e ordenamento dos bytes. Em outra opção convertem-se os dados em um formato neutro, por exemplo *big-endian*, para sua posterior adaptação, se necessário. Finalmente a máquina que serializa os dados pode indicar, dentro do *stream*, o tipo de ordenamento escolhido, normalmente o mesmo que usa na máquina internamente.

Na Figura 2.1 mostra-se graficamente a questão do *marshalling/unmarshalling*, na qual considera-se um exemplo de *marshalling* usando o formato CDR (*Common Data Representation*) especificado pela OMG no documento do CORBA 2.3 [CORBA99]

No caso da linguagem Java, utiliza-se sempre o sistema *big-endian* para armazenar as primitivas em um *stream*, resolvendo-se assim a questão da portabilidade dos dados.

Além do problema do ordenamento dos bytes, em um nível de abstração superior, existe também a questão do ordenamento dos componentes de um dado com uma estrutura complexa definida pelo programador, por exemplo, uma estrutura com vários campos de dados de diferentes tipos. Também aqui, deve-se definir como ordenar esses dados no *stream*, para que sua recuperação e carregamento em memória de sistemas diferentes sejam tais que a semântica e consistência dos dados não mude para a aplicação.

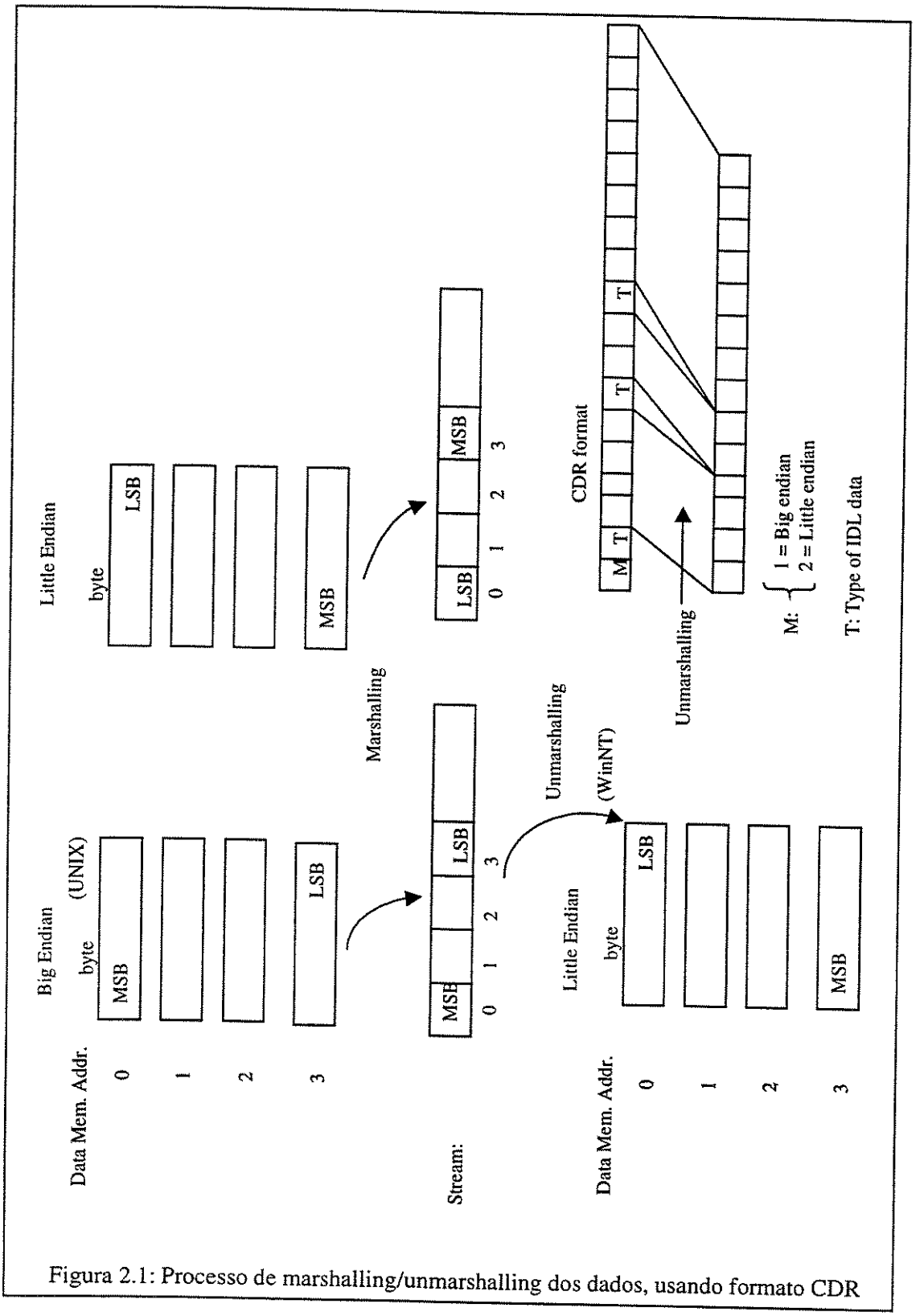

Figura 2.1: Processo de marshalling/unmarshalling dos dados, usando formato CDR

Um problema adicional ocorre quando os dados são trocados entre aplicações em linguagens diferentes, o qual pode acontecer em um ambiente CORBA. Neste caso deve

fazer-se o mapeamento adequado dos dados para que não haja perda de consistência. Com este objetivo, a OMG especificou uma sintaxe de transferência, denominada *Common Data Representation* (CDR) que permite o mapeamento dos dados entre as diferentes linguagens e onde a origem define se transmite em ordem *big-endian* ou *little-endian*.

Em resumo, o armazenamento de dados de um programa em um *stream* para que sejam transportados e recuperados em máquinas com CPUs e/ou sistemas operacionais diferentes, incluindo aplicações com linguagens diferentes, faz necessário arrumar os componentes desses dados numa forma tal que permita sua recuperação e carregamento sem perder sua consistência. Esses componentes, dependendo do nível de abstração, podem ser *bytes* abordando a questão do tipo de ordenamento dos *bytes* e de seu mapeamento para uma linguagem alvo ou, em um nível superior, abordando a questão de ordenamento dos componentes de um dado complexo. Este processo de ordenamento dos dados é conhecido na literatura [Cullens95] como ***marshalling***.

2.3 Paradigmas de mobilidade

Com os mecanismos de mobilidade disponíveis, é possível gerar modelos que possam permitir às aplicações tirar proveito desta mobilidade, sendo uma alternativa interessante aos modelos tradicionais. Ou seja, os programadores poderão contar com ferramentas adicionais para melhorar o desempenho de suas aplicações no ambiente atual das redes, gerando aplicações onde os atuais modelos são inadequados e, provavelmente, facilitando o esforço de programação para este tipo de ambientes distribuídos, onde o dinamismo e tamanho das redes são características relevantes. Isto não significa que modelos baseados na mobilidade venham a substituir os já existentes, mas sim que os complementem para adequá-los às exigências que os atuais ambientes estão impondo às aplicações.

Como modelos de programação que fazem uso da mobilidade do código [Fuggetta98] [Bieszczad98b] [Halls97] identificam-se os seguintes três paradigmas:

- **Avaliação Remota**: Este modelo consiste em enviar o código e os dados com os quais o código vai operar, quando seja executado no destino. O resultado será devolvido à origem. É utilizado, por exemplo, para aproveitar a capacidade de processamento de uma máquina remota. Diferencia-se do RPC (*Remote Procedure Call*) porque o código não se encontra permanentemente na máquina alvo, e só temporariamente, durante a sua execução. Desta forma, as aplicações que precisem de maior capacidade de execução podem fazer uso de máquinas que não estejam disponíveis no hospedeiro, onde se encontra a aplicação. Por outro lado, permite-se que a máquina de alto desempenho não fique sobrecarregada de código que não esteja sendo utilizado ou que é usado esporadicamente, poupando-se recursos do sistema. A avaliação remota baseia-se no mecanismo de cópia do código.
- **Código Sob Demanda**: Neste modelo, o destino solicita o código à origem, à medida que precise dele [Bic97]. Por exemplo os *applets* em Java operam desta

forma. Este modelo tem a vantagem de permitir atualizar dinamicamente as aplicações ou de adquirir ou alugar potencialidades que não tenham (por exemplo, para visualizar uma figura com um novo formato, solicita-se o manipulador desse tipo de formato).

- **Agentes Móveis** [Hohl97]: Este modelo consiste de software com autonomia para se movimentar pela rede, para realizar algum trabalho, em representação de um usuário (pessoa ou outro software). Este é um paradigma que acrescenta ao agente uma nova habilidade, além das já concebidas, reforçando ainda mais sua autonomia. Comparando-o com os modelos anteriores, representa o maior nível de abstração.

2.4 Os agentes móveis

Para definir o termo do agente móvel, primeiro deve-se definir o que é um agente. Existem diversas definições e não há um consenso geral, principalmente porque está sendo tratado por duas grandes linhas de pesquisa: Multi-agentes/Inteligência Artificial [Petrie96] e Engenharia de Sistemas.

No que diz respeito a este trabalho, ele está inserido na área da Engenharia de Sistemas, na qual um agente é considerado como uma entidade de software que realiza um trabalho em representação de seu dono, que pode ser um usuário ou uma aplicação.

Assim um agente móvel é um agente que tem a habilidade de movimentar-se numa rede, dentro de um domínio onde tenha condições para tal e esteja autorizado para desenvolver essa habilidade.

Em geral, o agente pode ter outras habilidades como: inteligência, autonomia, comunicabilidade, cooperação, confiabilidade e representabilidade.

Na realidade, para os pesquisadores ainda não existe unanimidade sobre a utilidade do paradigma do agente móvel: existem aqueles que o apóiam como outros que duvidam de suas vantagens. Mas alguns aspectos justificam considerar e analisar o paradigma do agente móvel como um modelo alternativo aos já existentes, com grandes potencialidades de uso:

- **Melhoramento do desempenho das aplicações nas redes de comunicação:** O tamanho das redes está crescendo rapidamente, não somente a nível global, como a Internet, onde empresas e usuários simples estão-se comunicando para participar de atividades como comércio eletrónico, telecomunicação, etc., mas também nos níveis corporativos, como intranet e Extranet. Com isso, o aumento do tráfego e o aumento da informação disponível em diversas estruturas estão afetando negativamente o desempenho das redes. Por outro lado, a interação

entre as aplicações e os usuários está sendo feita na sua maioria com o uso do paradigma do modelo cliente/servidor, o qual não está ajudando muito a resolver esse problema. No curto e no médio prazos não parece viável uma solução de tecnologia de rede que permita manter um bom desempenho da comunicação, com o crescimento exponencial dos usuários.

- **Satisfazer novos requerimentos:** Independentemente do cenário anterior, o desenvolvimento tecnológico está permitindo oferecer aos usuários serviços cada vez mais sofisticados e úteis, abrangendo necessidades que antes eram satisfeitas fora das redes, como também a criação de novos serviços. Isto significa que novos requerimentos são impostos às aplicações em rede, para que elas possam ser aceitas com sucesso no mercado. Destes requerimentos podem se distinguir os seguintes:
  - **Mobilidade:** Os usuários comunicam informação ou têm acesso a sistemas de informação, mesmo em movimento.
  - **Flexibilidade:** O suporte à mobilidade não é realizável se os sistemas não forem suficientemente flexíveis, para se adaptarem às novas situações ou ambientes. Por exemplo, um usuário poderá querer acessar a informação através da rede, a partir de lugares geográficos diferentes e com diferentes dispositivos.
  - **Simplicidade:** Estes novos produtos ou aplicações devem ser oferecidos de forma simples aos usuários, mesmo que complexos no seu desenvolvimento e instalação. O usuário preocupa-se cada vez menos com a instalação das aplicações, pressupondo que a infraestrutura de processamento e comunicação o faça de forma transparente.
  - **Poupar recursos:** Através da rede, os provedores oferecerão recursos aos clientes na medida e no instante de tempo necessários. Desta forma, o cliente não carregará no seu computador aplicações completas que nunca usará totalmente.
- **Aplicações afins:** Existem aplicações [Papaioannou99] [Cai96] que, pela sua natureza, poderiam ser desenvolvidas mais facilmente utilizando o paradigma do agente móvel, ainda que não necessariamente precisem operar em rede. Por exemplo, a simulação de comportamento emergente, como é o caso do estudo do comportamento coletivo das formigas e bactérias, porque permite liberar objetos com autonomia de acordo com o comportamento programado [Twhite99]. No caso de sistemas tolerantes a falhas, onde às vezes os processos devem ser movimentados de um hospedeiro para outro, por causa de um falha em um deles, de acordo as pesquisas que estão sendo feitas, a utilização de paradigmas baseados na mobilidade parece ser mais natural e flexível de aplicar que os modelos tradicionais, permitindo diminuir a complexidade do seu desenvolvimento.

As razões anteriores permitem justificar o estudo para o desenvolvimento de plataformas que suportem aplicações baseadas em agentes móveis e determinar se realmente este

paradigma ajuda a resolver os atuais e futuros requerimentos, de acordo com os diferentes aspectos descritos.

Mas o anterior não será suficiente se o novo paradigma somente for usado em âmbito restrito. O seu sucesso dependerá muito do seu uso num escopo global, onde possa ser testado todo seu potencial. Para isso, é necessário desenvolver plataformas de domínio público, que permitam rapidamente a introdução de agentes, independentemente da sua origem. Os aspectos de portabilidade e de padronização terão um papel importante.

2.5 Requerimentos gerais das plataformas para o suporte de agentes móveis

A utilização do modelo do agente móvel gera problemas que antes não existiam e que devem ser resolvidos.

O fato que um agente está se movimentando através da rede, significa que vai visitar diversos tipos de ambientes, desde aqueles, onde todas as máquinas são de um mesmo tipo, até ambientes totalmente heterogêneos, onde as máquinas são de marcas e sistemas operacionais diferentes. Portanto, a implementação dos mecanismos de mobilidade implicará na solução de problemas fundamentais [Bic99] como os de portabilidade do código que está sendo transportado, da representação dos dados na memória do computador, da manipulação do estado, e da interoperabilidade (comunicação) [Bradshaw96] dos sistemas que suportam a mobilidade. Além destes, outros problemas podem surgir como identificação e gerenciamento do agente, segurança, e comunicação entre os agentes móveis com os agentes estáticos.

Estes problemas nem sempre estarão presentes por serem dependentes do tipo da aplicação. Por exemplo, se a aplicação vai se executar dentro de uma *intranet* de um departamento ou empresa, provavelmente a questão de segurança não é relevante. Se o agente somente vai ser transmitido da origem para um destino, sem continuar para outros lugares, então a questão de identificação é irrelevante. Se os agentes que se movimentam não precisam de comunicação entre eles, então essa questão também será secundária.

Assim, qualquer plataforma que suporte agentes móveis deve considerar os seguintes aspectos:

- Ter um sistema de diretório de nomes, para que o agente possa ser registrado e gerenciado durante toda sua vida.
- Oferecer um sistema de comunicação entre agentes, considerando que eles estão-se movimentando permanentemente.

- Oferecer facilidades para que o agente possa ser criado e gerenciado através de mecanismos de instalação e execução automática;
- Prover mecanismos de segurança para proteger os agentes dos sistemas que visita e vice-versa.
- Para maior portabilidade, os sistemas que suportam os agentes deverão ser interoperáveis e oferecer diversos ambientes de execução para os agentes.
- Operar de forma assíncrona e assim permitir que o proprietário do agente possa ligar/desligar seu computador quando quiser: entretanto o agente poderá permanecer ativo na rede fazendo seu trabalho.
- Oferecer um sistema para o transporte dos agentes.
- A plataforma deve ser de arquitetura aberta para permitir que: os serviços dos diversos fabricantes possam interoperar e para que os agentes de diversas origens possam interoperar entre si e com o sistema.
- A plataforma deve ser suficientemente flexível para que possa ser ampliada e possa oferecer novos serviços, caso novos tipos de agentes devam ser suportados.

Estes aspectos levam a definir uma arquitetura geral de uma plataforma para o suporte dos agentes móveis, que será denominada Agência ou Servidor de Agentes, identificando-se os

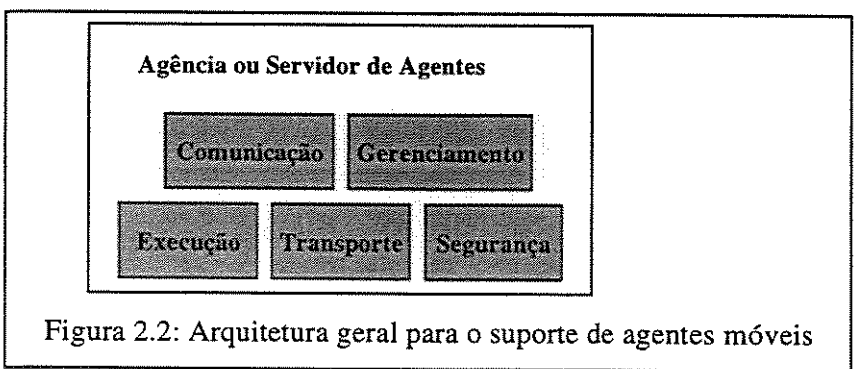

Figura 2.2: Arquitetura geral para o suporte de agentes móveis

seguintes módulos funcionais, como se apresenta na Figura 2.2.

- **Comunicação:** O ambiente deve fornecer facilidades para que os agentes, seus donos e sistema que os hospedam, possam se comunicar, de forma transparente e eficiente.
- **Transporte:** O ambiente deve oferecer serviços que permitam transportar um agente de um lugar para outro.
- **Execução:** O ambiente deve oferecer facilidades para que os agentes recebidos possam ser instalados, executados e removidos corretamente.
- **Segurança:** O ambiente deve garantir níveis de segurança, tanto para os agentes como para o sistema que os hospedam. Estes níveis de segurança consideram os aspectos de comunicação, transporte e execução.

- **Gerenciamento:** O ambiente deve oferecer as facilidades para administrar o ciclo de vida do agente através de toda a rede onde se está movimentando, desde sua criação até sua eliminação.

2.6 Escopo do presente trabalho

Baseado no anterior, corresponde agora definir o contexto do presente trabalho, o qual tem como objetivo fundamental estudar os problemas relacionados com a mobilidade dos agentes e propor uma arquitetura orientada a objetos que permita suportar esta facilidade.

A facilidade considerará, total ou parcialmente, as funcionalidades já identificadas como se explica na continuação:

- O transporte será considerado na sua totalidade, por estar diretamente relacionado com a mobilidade.

- Em relação a comunicação, será considerada a relação entre o agente e o serviço que permite usar a facilidade para se movimentar, além da comunicação entre os ambientes distribuídos na rede que suportam a facilidade.
- Só os aspectos de segurança no transporte serão considerados.

- Não serão considerados os aspectos relacionados com a execução dos agentes, mas será identificada sua relação com este módulo.

- Somente aspectos de registro dos agentes serão considerados no módulo de gerenciamento.

2.7 Aspectos relacionados com o transporte dos agentes

Até agora, tem se tratado os agentes móveis sob um ponto de vista geral. Cada um dos requerimentos apresentados poderia ser abordado em maior profundidade. Contudo o presente trabalho vai se concentrar só nos problemas que devem ser resolvidos no transporte dos agentes, com a proposta, no capítulo 5, das funcionalidades que um serviço desse tipo deve possuir.

Existe atualmente uma variedade de plataformas que suportam agentes móveis, algumas delas orientadas para um certo tipo de aplicações e para ambientes de execução e sistemas operacionais homogêneos, outras para sistemas operacionais heterogêneos (Java). Todas elas abordam a questão do transporte dos agentes de forma diferente, ficando estes sistemas

isolados uns dos outros. Seria, portanto, adequado desenvolver uma plataforma que permita transportar os diversos tipos de agentes utilizando uma arquitetura aberta. Isto permitiria a interoperabilidade dos diversos sistemas e o aumento da eficiência no uso dos recursos da rede.

Considerando só o transporte de agentes, determinou-se que os seguintes aspectos devem ser abordados quando se trata de desenvolver um serviço correspondente.

2.7.1 Representação dos dados

O agente movimenta-se junto com dados que ele mesmo gera ou recebe. No caso de agentes implementados com linguagens compiladas, podem acontecer problemas de representação dos dados em máquinas diferentes. Dependendo da máquina e/ou sistema operacional, os dados podem ser representados em memória no modo *Big-endian* ou *Little-endian* como já foi explicado anteriormente. Por outro lado, como os dados devem ser transportados, eles deverão ser salvos serialmente em um arquivo e logo recuperados corretamente no destino, sem perda de semântica. Portanto, o processo de *marshalling/unmarshalling* deverá ser considerado como uma atividade adicional, prévia à transmissão/recepção do agente com seus dados.

2.7.2 Interoperabilidade dos sistemas

Para que a movimentação do agente aconteça, é necessário que o sistema que suporta os mecanismos de mobilidade se comunique com os sistemas homólogos no destino, para realizar as tarefas que permitam negociar e transferir o agente em conjunto com outras informações opcionais, e de forma transparente para a aplicação. Portanto deve existir comunicação entre estes sistemas, alojados em máquinas heterogêneas, fazendo-se necessário o uso de protocolos de comunicação conhecidos pelas partes envolvidas e pelo ambiente de rede em que eles operam. Por exemplo, em redes intranet e Internet serão adequados protocolos baseados em TCP/IP, como RPC, HTTP, e IIOP.

2.7.3 Identificação dos agentes e das agências

Em geral, um programa identifica-se univocamente dentro de uma máquina pelo nome de arquivo e pelo nome do diretório e sub-diretórios no qual foi instalado. No caso do agente móvel, este tipo de identificação não será mais válida, devendo buscar-se outra solução que permita identificar o agente univocamente, não só em uma máquina, mas também na rede. Alguns autores [Lingnau95] [Lugmayr99] definem um agente como móvel, quando o agente possui identificação persistente, para salientar que sua identificação deve permanecer durante toda a sua a vida, o que implica no desenvolvimento de uma forma de diretório ou de registro na rede para suportar este mecanismo, de forma transparente para a aplicação.

2.7.4 Empacotamento do agente

Como se falou anteriormente, o agente é uma entidade de software que pode ser composta, além do código, de outros dados como: arquivos descritores do agente, arquivos com o estado do agente, informação para a instalação, propriedades, etc. Todos estes arquivos terão que ser despachados para o destino. Portanto, deve-se desenvolver um esquema para implementar um receptáculo que inclua um cabeçalho de controle para ordenar flexivelmente os diferentes componentes do agente, de tal forma que, nem sempre, todos os componentes tenham que ser parte do pacote que está sendo enviado. Por exemplo, pode acontecer que os arquivos que contêm o código do agente não sejam enviados a partir do lugar atual do agente, mas sim a partir de um servidor; ou que o componente correspondente ao estado do agente não seja enviado por se tratar de uma cópia.

Como a transmissão pode ser feita usando qualquer tipo de protocolo de transporte (síncrono ou assíncrono), será necessário que uma identificação do agente seja incluída no pacote para que este possa ser identificado no destino.

Além disso, como a informação de controle vai ser processada por sistemas diferentes, haverá que considerar os problemas associados com a representação dos dados, como já foi explicado anteriormente. Logo, no cabeçalho de controle, deverão ser incluídos mecanismos que permitam a decodificação correta dos dados correspondentes à informação de controle.

Portanto, é necessário que o cabeçalho de controle contenha a seguinte informação:

- Nome do agente
- Tipo de ordenamento que está sendo utilizado pelos dados numéricos.
- Se vão se colocar *strings* de caracteres, haverá que indicar se o caractere está sendo representado por um ou dois bytes.
- Tamanho e ordem dos arquivos que compõem o agente.
- Versão do formato do empacotamento que está sendo utilizado.
- Opcionalmente, inclusão da informação referente à assinatura e credenciais do agente.

2.7.5 Negociação

Se algum tipo de consulta ao destino está sendo considerada, como passo prévio à transmissão do agente, então deverá ser definido um protocolo para a negociação, entre outros, dos seguintes aspectos:

- Tipo de protocolo de transporte para transmitir o agente.
- Se o código do agente será transmitido a partir do lugar atual do agente ou a partir de um servidor, indicando-se seu endereço.
- Ambiente de execução que o agente utiliza (sistema operacional, CPU, memória necessária).
- Se o código do agente deve ser compilado, interpretado ou executado diretamente.

O protocolo de negociação deve ser tal que não gere uma sobrecarga muito grande para o serviço de transporte. Isto significa que deverão ser avaliados diversos mecanismos para permitir uma negociação simples e expedita, por exemplo, trocando dados utilizando uma estrutura de dados predefinida ou trocar um documento que contenha comandos predefinidos.

2.7.6 Segurança no transporte

A única questão de segurança associada à transmissão do agente tem relação com a proteção do conteúdo do pacote contra terceiros que tentem abri-lo e decodificar a informação nele contida, para alterá-la ou para tirar proveito dela. Este é um problema já existente nas redes, o qual normalmente é resolvido utilizando mecanismos diversos de criptografia com chaves de segurança públicas ou privadas. Então, se necessário, uma vez construído o pacote, este terá que ser criptografado, por exemplo, utilizando-se chaves obtidas durante a negociação, ou por algum outro mecanismo, antes de ser transmitido ao destino.

2.8 Resumo

Como se descreveu neste capítulo, a mobilidade de código foi tratada a partir de três abordagens diferentes: a nível do sistema operacional, a nível da linguagem de programação e a nível da camada intermediária. No presente trabalho, vai-se considerar a abordagem a nível da camada intermediária.

Foi feita uma classificação dos mecanismos de código móvel, considerando-se quem toma a iniciativa da transferência (*fetch* e *forward*) e considerando os mecanismos de mobilidade (cópia, movimento e migração). Também definiram-se os conceitos de estado e de *marshalling*, os quais estão associados à mobilidade do código, e descreveram-se vários modelos de mobilidade, como avaliação remota, código sob demanda e agentes móveis.

Justificou-se a análise do paradigma do agente móvel, destacando-se as principais potencialidades que este modelo oferece: melhoramento do desempenho das aplicações nas redes de comunicação, garantia de novos requerimentos das aplicações na rede e afinidade do modelo com certos tipos de aplicações.

Finalmente, foi feita uma análise geral para determinar a arquitetura de uma plataforma para o suporte de agentes móveis. A partir desta arquitetura definiu-se, como escopo do presente trabalho, o transporte de agentes, descrevendo-se as principais questões associadas (representação dos dados, interoperabilidade dos sistemas, identificação dos agentes e das agências, empacotamento do agente, negociação e segurança no transporte).

3 Plataformas para o suporte da mobilidade

Neste capítulo faz-se uma análise das plataformas sobre as quais se poderia construir um ambiente para o suporte dos agentes móveis. Achou-se CORBA e Java como as ferramentas mais adequadas para construir estas plataformas: as razões que justificam esta decisão serão discutidas ao longo deste capítulo. Sob o ponto de vista da mobilidade, primeiramente serão analisados os serviços CORBA e posteriormente as potencialidades da linguagem Java.

3.1 Mobilidade em CORBA

Nesta parte far-se-á uma descrição e análise da documentação CORBA da OMG, que tem direta relação com os aspectos da mobilidade de um objeto. Com relação à especificação CORBA 2.3 descrever-se-ão as mensagens GIOP (*General Inter-ORB Protocol*), que dão suporte à mobilidade de objetos, e o POA, (*Portable Object Adapter*), que dá suporte à portabilidade dos objetos Corba. Também serão descritos os serviços CORBA: *LifeCycle*, *Externalização* e Nomes, bem como a Facilidade de Sistemas de Agentes Interoperáveis (MASIF: *Mobile Agent System Interoperability Facilities*) que já foi aprovada, estando em fase de implementação.

3.1.1 Serviço de LifeCycle

Este serviço [LifeCycle97] permite controlar a vida de um objeto desde sua criação, incluindo sua cópia, movimentação e "morte". Este serviço oferece três interfaces, reunidas no módulo nomeado *CosLifeCycle*:

- *FactoryFinder*, para procurar por um tipo de objeto;
- *GenericFactory*, para criar objetos, sugerindo diferentes estratégias para usá-la;
- *LifeCycleObject*, para eliminar, copiar e mover objetos.

Além destas interfaces, acrescentam-se outras para gerenciar o ciclo de vida de conjuntos de objetos relacionados (grafos), reunidas no módulo *CosCompoundLifeCycle*, cujas interfaces estão relacionadas pela herança, com as interfaces do serviço de *Relationship*.

No que diz respeito a este trabalho, só vai-se considerar o aspecto da mobilidade para sua posterior análise, com o fim de determinar se é adequado para plataformas que suportem agentes móveis.

Em primeiro lugar, para que a vida de um objeto possa ser controlada, usando-se este serviço, é necessário que o objeto implemente a interface *LifeCycleObject*; o resto das interfaces não são parte do objeto. Isso significa que o programador encarregado de

implementar o objeto deverá preocupar-se em implementar as operações (*copy, move* e *remove*) que constituem esta interface.

3.1.1.1.Movimentação de objetos

A movimentação dos objetos é feita utilizando a operação *move* da interface *LifeCycleObject*. No diagrama da figura 3.1 apresenta-se como acontece a movimentação de um objeto desde um lugar para outro. Antes de começar a explicação, descreve-se cada um dos objetos que participam na interação.

- **"ObjectX"**: É o objeto que se quer movimentar de um lugar para outro. Tem o nome de "ObjectX" e é um objeto permanente; isto é, pode ser encontrado através de um serviço de *Naming* ou *Trading*. Por outro lado implementa o conjunto das operações definidas pela interface *LifeCycleObject* do serviço *LifeCycle*.
- ***FactoryFinder***: É a classe que implementa a interface *FactoryFinder* do serviço *LifeCycle*. Neste caso temos uma instância dela (objeto) nomeada "FactoryFinder1". Possui a operação *find_factories*, para procurar por um fabricante de objetos do tipo indicado no parâmetro de entrada desta operação. A procura pode ser feita usando os serviços de *Naming*, *Trading* ou quaisquer outros similares.
- ***GenericFactory***: É a classe que implementa a interface *GenericFactory* do serviço *LifeCycle*. No exemplo temos uma instância dela nomeada "ObjectXFactory". Possui a operação *create_object*, a qual cria um objeto do tipo equivalente ao tipo do objeto "ObjectX" e o ativa, ficando pronto para receber invocações e devolvendo uma referência de um objeto básico ou *root* nomeado *Object*. Logo, o invocador deverá aplicar a operação *narrow* sobre a referência para garantir que vai usar um objeto do mesmo tipo desejado, uma vez que se utilizou uma fábrica genérica de objetos.No lado cliente especifica a devolução da referência a um objeto do tipo *Object* (objeto raiz de todos os objetos CORBA). No exemplo não é necessário aplicar o *narrow*, porque só é importante criar o objeto no destino.
- ***Client***: É a aplicação que deseja movimentar o objeto "ObjectX" de um local para outro, e do qual tem a referência.

Seqüência para movimentar um objeto:

1. O cliente invoca a operação "move" do "ObjectX", entregando como parâmetro uma referência para o objeto "FactoryFinder1", da classe *FactoryFinder*, que fisicamente deve estar no lugar de destino. Ou seja, de alguma forma o cliente deve saber ou possuir a referência de um objeto *FactoryFinder*, o qual deve estar no lugar de destino.

2. A partir de agora, a operação *move* controlará a movimentação do objeto "ObjectX". Ela começa a invocar a operação *find_factories* do objeto "FactoryFinder1". Esse objeto procurará por um fabricante de objetos do tipo equivalente ao "ObjectX". Após encontrá-lo, devolve a referência do objeto fabricante; neste exemplo foi achado um objeto nomeado "ObjectXFactory".
3. Como agora o cliente tem uma referência para um objeto que fabrica objetos do tipo desejado, invoca a operação *create_object* do "ObjectXFactory", para criar o objeto que se quer movimentar e devolvendo a referência do objeto criado.
4. Depois que o fabricante cria o objeto, procede à sua ativação, ficando o objeto pronto para receber invocações.
5. Em seguida o objeto "ObjectX" é eliminado (desativado) no lugar inicial, por que já existe outro idêntico no lugar de destino.
6. O cliente passa a invocar operações sobre o objeto *Object*. Como agora está registrado em outro lugar, o ORB se encarrega de procurá-lo na rede obtendo uma referência dele para o cliente.

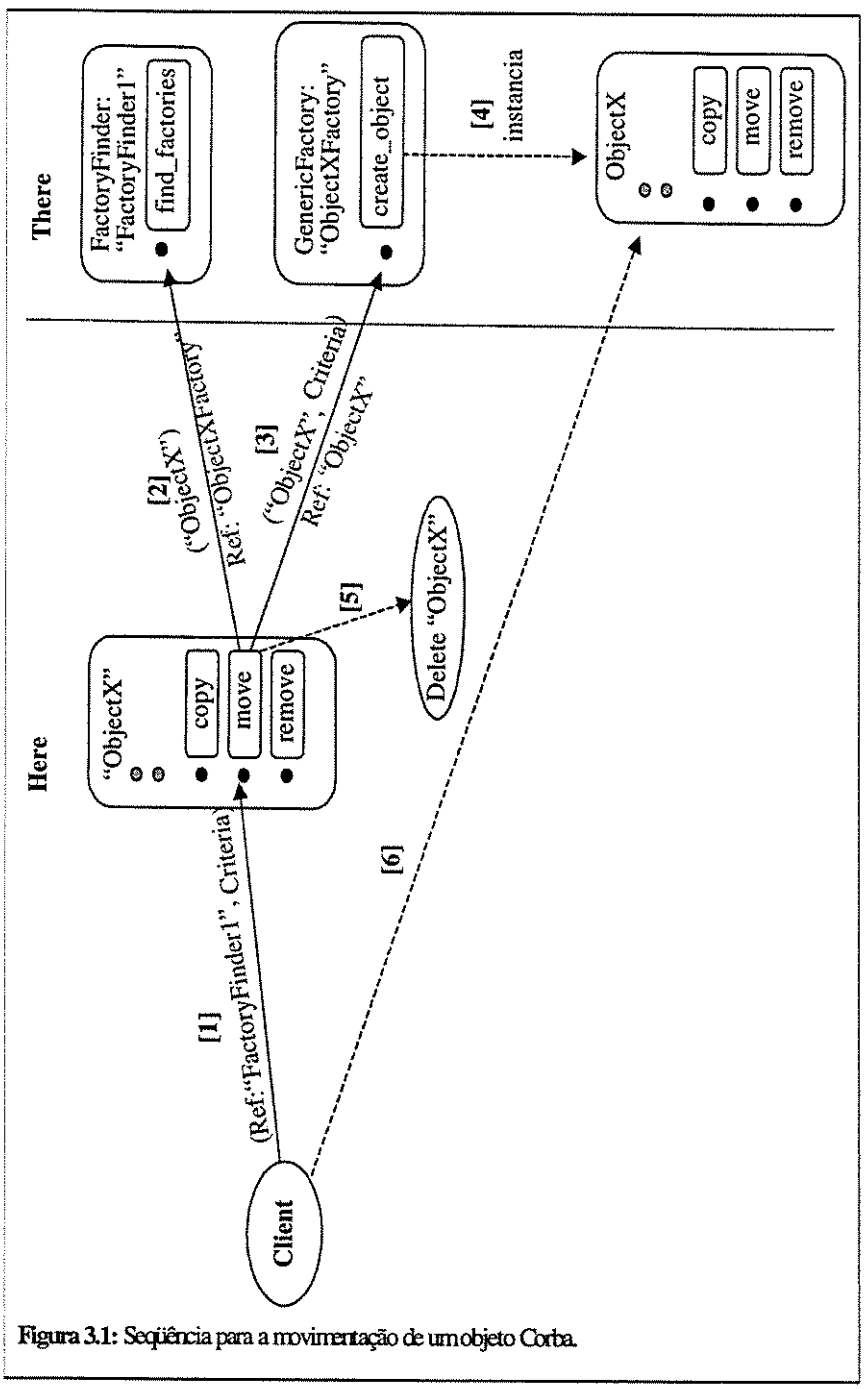

Figura 3.1: Seqüência para a movimentação de um objeto Corba.

Em função dos fatos explicados podem fazer-se as seguintes observações:

- O serviço assume que, no destino, deve existir um fabricante de objetos do tipo desejado. Logo, cabe ao programador estabelecer como o fabricante de objetos será transportado até ali ou se tem fabricantes espalhados pelos lugares que o cliente vai percorrer.
- As operações *move* e *copy* diferem uma da outra porque nesta última, após ser executada, o objeto original não é eliminado e o cliente terá duas referências diferentes para objetos do mesmo tipo, mas que estão em lugares diferentes. Na operação *move*, o cliente terá sempre uma referência para um mesmo objeto que se movimentou de forma transparente para ele. Por tal motivo, a operação *move* não devolve nenhum parâmetro. O cliente continua utilizando a mesma referência antes e depois de executar-se a operação *move*.
- A interface *LifeCycleObject*, especificada pelo serviço *LifeCycle*, deve fazer parte do objeto alvo. Ou seja, o programador do objeto deve-se preocupar com esta interface.
- Um objeto especializado em procurar objetos fabricantes deve suportar a interface *FactoryFinder* do serviço *LifeCycle*, para que ele possa ser utilizado pelos clientes do sistema CORBA, de forma padronizada.

Portanto, se este serviço for usado para implementar o mecanismo de movimentação de objetos, devem considerar-se os seguintes aspectos:

(1) O serviço de *LifeCycle* não especifica nada sobre como transportar um objeto quando se utiliza a operação *move*. Só assume que o fabricante do objeto deve estar no destino, para que seja encontrado pelo *factory finder* e seja utilizado para instanciar o objeto no local.

(2) A operação *move* é parte da interface do objeto. Se o objeto é transportado, significa que o código da operação *move* também é transportado junto com o resto do código.

(3) Quando se movimenta um grupo de objetos, o cliente só precisa mover o primeiro objeto e o serviço moverá o resto de forma transparente. Evidentemente, o programador deste serviço deverá providenciar a forma de fazê-lo.

(4) A operação *move* é um processo síncrono. Isto significa que o cliente não poderá fazer nada até que a execução da operação termine.

Considerando os fatos anteriores pode se concluir o seguinte:

- Considerando o item 1 anterior, quando o agente visitar lugares pela primeira vez, é necessário implementar um mecanismo de transporte para mover o agente como um

grupo de objetos. Também será necessário implementar um mecanismo de fabricação para instanciar o agente no destino. Outra questão para considerar, é o fato de não ser necessário implementar um mecanismo de busca de fabricantes de agentes, porque o agente vai ser transportado para o destino com seu fabricador incluído e portanto, vai estar ali antes de ser instanciado.

- Do ponto de vista de desempenho e considerando os itens 2 e 3 anteriores, pode se dizer que não é adequado usar a operação *move* do serviço *LifeCycle* para suportar a mobilidade dos objetos, porque significa sobrecarregar o código do agente, demorando mais seu transporte. Se o controle da movimentação do agente estivesse na plataforma ter-se-iam as seguintes vantagens, melhorando o desempenho da aplicação:

  1. Um objeto mais leve: O código que implementa a operação *move*, já não estaria em cada objeto que compõe um agente, senão só uma vez na plataforma. Além disso, o programador do agente não se preocuparia com a implementação desta operação e só como usá-la.

  2. Se tivermos uma entidade formada por um grupo de objetos (por exemplo, um agente), será possível implementar um algoritmo mais eficiente para mover o grupo inteiro e não objeto a objeto.

- Do item 4, pode-se dizer que não é adequado fornecer um serviço síncrono porque o processo de transporte pode levar um tempo excessivo, convertendo-se em uma desvantagem para a aplicação que espera pela finalização do serviço.

Considerando estes fatos, decidiu-se propor, nos capítulos posteriores deste trabalho, um serviço diferente que permita o transporte de agentes, o qual formará parte de um conjunto de serviços que nomearemos Serviço de Facilidades para Agentes Móveis.

3.1.2. Serviço de Externalização

Este serviço [Externaliz97], conhecido como externalização, permite que um objeto possa salvar o valor de seus atributos (externalizar), ou seja "salvar" seu estado, , em um meio que facilite sua posterior recuperação ou internalização. A recuperação pode ser no mesmo lugar em que aconteceu a externalização ou em outro local, não sendo responsável este serviço pelo transporte do arquivo que contém os valores. O serviço especifica salvar valores de tipos básicos definidos por IDL, ou seja, não se pode especificar a externalização de um tipo de dados estruturado: neste caso deve-se salvar, por separado, cada dado que compõe a estrutura, sendo responsabilidade do programador controlar esta situação.

Este serviço abrange a externalização de grafos de objetos, portanto tem estreita relação com os serviços do *Relationships* além do *LifeCycle* para a criação de objetos. Mas neste trabalho só vai se explicar o referente à externalização de um objeto simples para mostrar o mecanismo básico deste serviço.

3.1.2.1.Processo de externalização

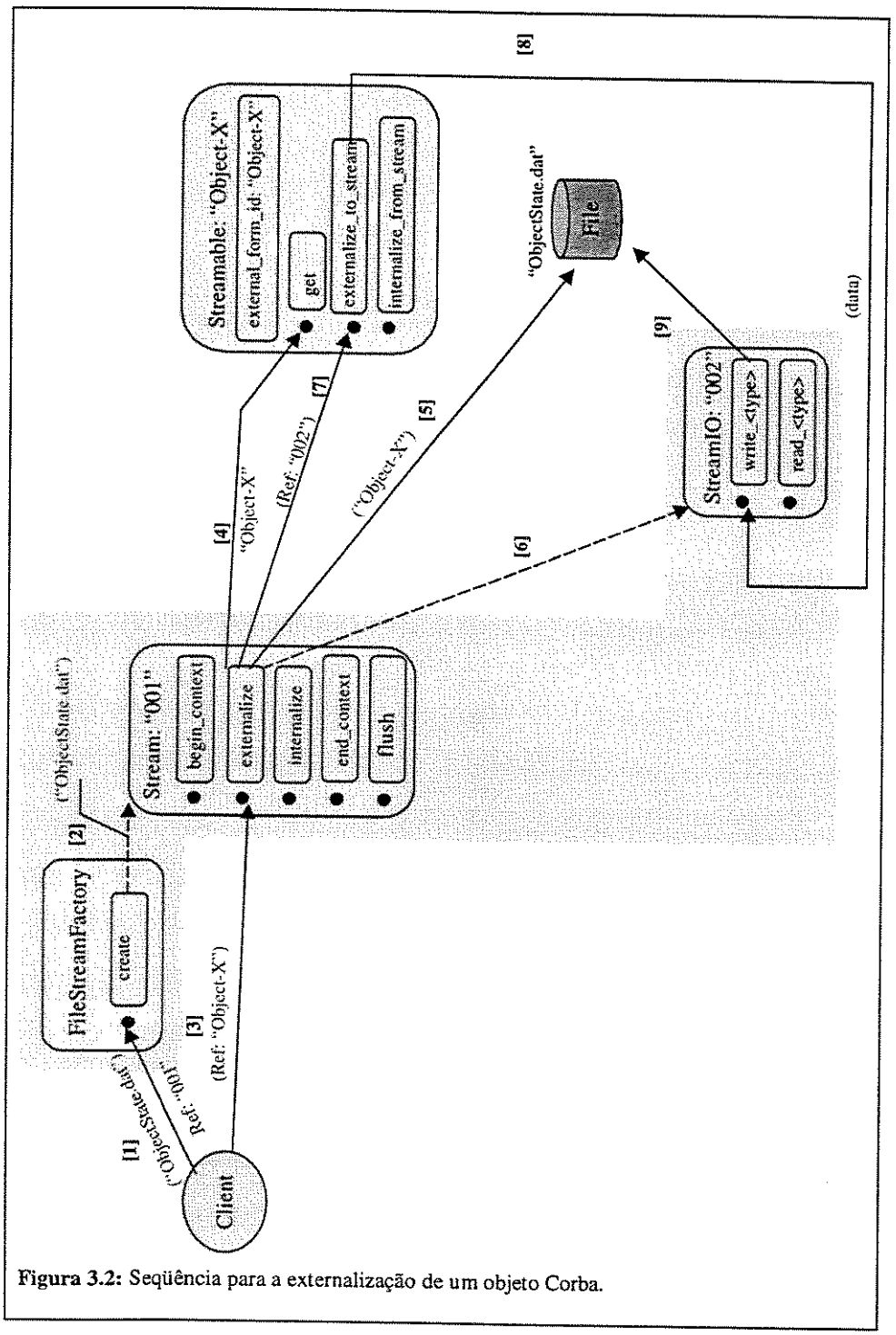

Figura 3.2: Seqüência para a externalização de um objeto Corba.

O processo de externalização de um objeto é apresentado na Figura 3.2.

Neste processo participam as seguintes quatro interfaces especificadas pelo serviço:

- ***Streamable*:** As operações especificadas por esta interface são parte do objeto alvo. Definem-se duas operações e um atributo, para gerenciar o processo de externalizar/internalizar o estado do objeto que é iniciado pelo cliente do mesmo. Como é parte do objeto alvo, o programador se deverá preocupar da sua implementação.No exemplo, o objeto alvo está identificado pelo nome "Object-X".
- ***FileStreamFactory*:** Esta interface especifica um objeto que permite criar a interface *Stream*. Em outras palavras, especifica um objeto que é registrado como permanente para que este serviço de externalização seja encontrado e utilizado. Neste caso, o cliente conhece "a priori" a referência a um objeto do tipo *FileStreamFactory*.
- ***Stream*:** Esta interface especifica as operações que controlam os processos de externalização e internalização, além de outras operações auxiliares. No exemplo, esta interface está representada por um objeto transiente nomeado "001".
- ***StreamIO*:** Esta interface especifica um conjunto de operações *write/read*, que permitem salvar/recuperar os dados em/desde um arquivo seqüencial no disco. Estas operações serão controladas pelo objeto alvo através da interface *Streamable*. Os dados serão salvos em *bytes* de acordo as regras de *mapping* entre IDL e a linguagem utilizada.

Seqüência para o processo de externalização:

1. Através de um objeto do tipo *FileStreamFactory*, o cliente solicita criar um objeto do tipo *Stream*, o que é feito através da operação *create* e repassando, como parâmetro, o nome de um arquivo ("ObjectState.dat") para que ali sejam salvos o estado do objeto. No retorno da operação, o cliente recebe uma referência a um objeto do tipo *Stream*.
2. O objeto fabricante cria um objeto do tipo *Stream*, nomeado "001" e repassa para ele um parâmetro com o nome do arquivo definido pelo cliente.
3. Com a referência conhecida, o cliente invoca a operação "externalize" do objeto "001", repassando, como parâmetro, a referência do objeto que quer externalizar.
4. Agora o objeto "001" solicita ao objeto alvo sua identificação ou nome, presente como um atributo do tipo *string*. No exemplo, o nome do objeto alvo é "Object-X". Isto é necessário, porque a identificação do objeto (e não sua referência) será salva como o primeiro dado no arquivo; desta forma identifica ao dono dos dados correspondentes ao estado desse objeto.
5. Dentro do arquivo "ObjectState.dat", procede-se a salvar o nome que identifica o objeto alvo.
6. O objeto "001" cria um objeto do tipo *StreamIO*, para que finalmente esse objeto sirva de intermediário para salvar os dados no arquivo.
7. O objeto "001" invoca a operação *externalize_to_stream* do objeto alvo, para proceder à externalização do objeto alvo.

8. O objeto alvo ("Object-X") procede a invocar as operações *write_<type>* de acordo com o tipo de dado IDL que quer salvar, até finalizar.

3.1.2.2.Processo de internalização

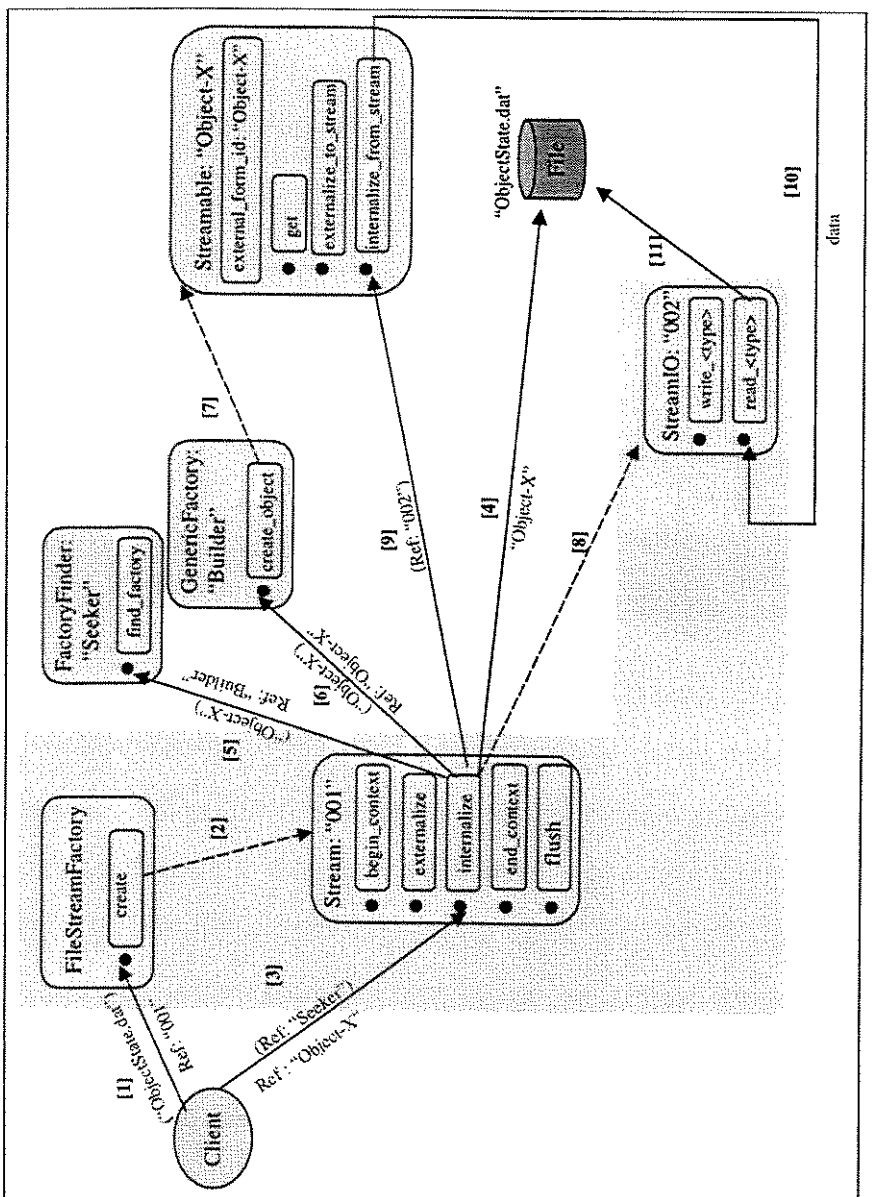

Figura 3.3: Seqüência para a internalização de um objeto Corba.

Para o processo de internalização, o qual é apresentado na Figura 3.3, além dos objetos anteriormente explicados, participam os objetos do tipo *FactotyFinder* e *GenericFinder* pertencentes ao serviço de *LifeCycle*, com o objetivo de criar um objeto que posteriormente será internalizado.

Seqüência para o processo de internalização: (assume-se que o objeto ainda não existe)

1. O cliente solicita a um objeto do tipo *FileStreamFactory* que crie um objeto do tipo *Stream*. Além disso, repassa o nome do arquivo (no exemplo, "ObjectState.dat") onde deve estar o estado do objeto que precisa internalizar. No retorno da operação, recebe a referência do objeto do tipo *Stream* criado.
2. O objeto do tipo *FileStreamFactory* cria um objeto do tipo *Stream*, nomeado "001".
3. O cliente invoca a operação *internalize* do objeto "001", entregando como informação a referência de um objeto (de nome "Seeker") do tipo *FactoryFinder*

(quem posteriormente buscará um fabricador de objetos do tipo que o cliente quer internalizar). Ao terminar a internalização, esta operação devolverá ao cliente a referência do objeto internalizado.

4. O objeto "001", como já conhece o nome do arquivo, procede a ler o primeiro valor que deve corresponder à identificação do objeto (no exemplo, "Object-X").
5. Agora o objeto "001" possui como informação a referência a um objeto do tipo *FactoryFinder* e o nome que identifica ao objeto que se quer internalizar. Logo, solicita a "Seeker" procurar por um objeto que possa criar um objeto cuja estrutura esteja associado ao nome "Object-X". De volta, recebe uma referência do criador, nomeado "Builder", de esse objeto.
6. O objeto "001" solicita a "Builder" que crie um objeto do tipo "Object-X". No retorno recebe a referência do objeto criado.
7. O objeto "Builder" cria um objeto com o nome "Object-X".
8. O objeto "001" cria um objeto do tipo *StreamIO*, nomeado "002". Durante a criação entrega a informação sobre o arquivo que deve manipular para quando leia os dados.
9. O objeto "001" invoca a operação *internalize_from_stream* do objeto alvo, para que inicie a internalização de seu estado. Para isso, entrega como parâmetro, a referência ao objeto "002", encarregado de executar a recuperação dos dados diretamente do arquivo.
10. Agora, o objeto realiza a internalização de seu estado, usando as operações *read_<type>* do objeto "002", na mesma seqüência em que foram externalizados.
11. O objeto "002" lê os dados a partir do arquivo seqüencial "ObjectState.dat", à medida que o objeto alvo o solicite através das operações de leitura. Terminado o processo de recuperação do estado, o controle retorna ao cliente, junto com a referência do objeto alvo que acabou de ser criado e internalizado.

3.1.2.3.O serviço de Externalização considerando os aspectos da mobilidade

A partir da descrição anterior, podem-se deduzir as seguintes considerações:

- O cliente pode criar um objeto do tipo *Stream* para que este último se encarregue de realizar a externalização. Isto pode ser feito de duas formas:
  - Usando a interface *StreamFactory*, que cria um objeto do tipo *Stream*, ou
  - Usando a interface *FileStreamFactory*, que além de criar um objeto do tipo *Stream*, permite que o cliente especifique o nome do arquivo onde os dados serão salvos.

    No exemplo, utilizou-se a segunda opção.
- O atributo *external_form_id*, o qual guarda o nome do objeto, é uma estrutura que tem dois membros, *id* **e** *kind*, ambos do tipo *string*. Mas só o membro ***id*** é

considerado. A leitura deste dado faz-se através da operação padrão para ler atributos de objetos distribuídos, nomeada *get*.

- O objeto do tipo *FileStreamFactory* é registrado como permanente para que possa ser localizado por algum meio privado ou padrão como, por exemplo, usando o serviço de *Naming* ou *Trading*, atuando como representante do serviço de Externalização. Ou seja, através deste objeto qualquer cliente poderá ter acesso a este serviço para externalizar ou internalizar um objeto.
- Os objetos dos tipos *Stream* e *StreamIO* ("001" e "002", no exemplo respectivamente), são criados como objetos transientes, cujas vidas vão durar os tempos necessários para executar a externalização ou internalização. Não é adequado que sejam criados como objetos permanentes porque consumiriam recursos, estando eles ociosos, além do que deveriam ter mecanismos de multitarefa ou *threading* para que sejam compartilhados pelos diferentes clientes. Aliás eles podem ser criados separadamente porque implementam interfaces diferentes. A especificação indica que ambas interfaces também podem ser implementadas usando um objeto só, o que seria mais eficiente, ficando a critério do desenvolvedor do serviço. No exemplo apresentado, criaram-se separadamente para uma melhor clareza na explicação.
- No exemplo, o serviço de *Stream* (através do objeto "001") criou o objeto alvo de acordo com os mecanismos-padrão especificados, ou seja, usando o serviço de *LifeCycle* (*FactoryFinder* e *GenericFactory*). Mas a especificação deste serviço estabelece que também é possível utilizar outras maneiras especificas para criar o objeto alvo.
- Como a interface *Stream* vai criar um arquivo (com o nome definido normalmente pelo cliente), que será usado pela interface *StreamIO*, a referência para este arquivo deve ser compartilhada por ambas interfaces. Sua implementação é uma questão que depende só do desenvolvedor do serviço.
- O serviço especifica um formato padrão para armazenar o estado como dados em um arquivo seqüencial, para permitir uma portabilidade dos dados através de diferentes implementações de CORBA e em máquinas com diferentes classes de CPUs. Em geral, o formato define o número de bytes por cada tipo de dado, de acordo com os tipos de dados definidos em IDL; além de definir um ordenamento *big-endian* para a escritura/leitura dos mesmos.
- O serviço também especifica como salvar/recuperar a referência de um objeto.
- Se o cliente quer salvar o estado de múltiplos objetos num mesmo arquivo, ele pode usar as operações *begin_context* e *end_context* controlando a interface *Stream* para salvar ordenadamente o estado dos diferentes objetos. A execução destas operações deve traduzir-se em marcas no arquivo, indicando o início e fim dos dados correspondentes ao estado de um objeto. Na realidade, o serviço especifica só o início.

Como resultado, pode-se concluir que, se o agente for um objeto CORBA ou um conjunto de objetos CORBA, pode-se utilizar este serviço para salvar o estado do agente.

Neste serviço, propõe-se um padrão que define o formato para salvar os diferentes tipos de dados definidos em IDL. Isto é vantajoso para os agentes móveis, porque permitiria transportar o estado do agente para lugares com plataformas diferentes sem perder a semântica dos dados.

3.1.3. Serviço de Nomes

Este serviço [Naming97] padroniza o formato que devem ter os nomes dos objetos Corba, definindo-se a seguinte estrutura denominada *Name*:

```
        typedef string Istring;
struct NameComponent {
                Istring id;
                Istring kind;
        };

        typedef sequence < NameComponent> Name;
```

Esta estrutura mostra que o nome de um objeto pode ser composto de uma série de componentes, onde cada componente é composto de duas partes, *id* e *kind*, a primeira parte a ser usada como nome e a segunda como identificador de tipo. O nome de um objeto Corba é definido como uma estrutura para não especificar a forma de separação de cada componente nem cada parte de um componente. A separação fica para a implementação local. Por exemplo:

/agencia-com/agencia_1

representaria um nome de dois componentes, onde o separador de componentes escolhido é o símbolo "/" e o separador de partes é o símbolo "-".

Através da operação *resolve*, definida na interface *NamingContext*, é possível determinar o endereço de um objeto entregando o nome e obtendo a referência de um objeto, como mostra o formato da operação:

```
Object resolve ( in Name n )
        Raises (NotFound, CannotProceed, InvalidName);
```

Atualmente a OMG está solicitando a modificação desta especificação nos aspectos seguintes:

- A conversão da estrutura dos nomes para string e vice-versa. Neste caso, está-se propondo uma sintaxe para representar o nome de um objeto como uma série de caracteres (*string*).
- Definir uma interface e mecanismos comuns, para a iniciação da aplicação com relação a obter a referência de um Serviço de Nomes ou outro serviço como o *Trading*. Para esta questão, estão-se propondo diferentes mecanismos, alguns deles agregando o conjunto de mensagens GIOP.

- Utilizar URL [URL94] como esquema alternativo para nomear objetos. Nesta questão, basicamente todas as propostas definem o formato seguinte:
  <nome do esquema>: //<endereço de máquina> <separador> <nome-string>
  onde:
  - <nome do esquema>: Propõem-se os seguintes valores: "ins" por Iona et al [IntNaming98a], "corbaname" por IBM et al [IntNaming98b].
  - <endereço de maquina>: Especifica a máquina onde se encontra um Serviço de Nomes ou outro como *Trading*. O formato é similar ao esquema URL padrão, ou seja, nome do domínio da máquina e número de porta (opcional).
  - <nome-string>: representa o nome do objeto do qual se busca a referência: o nome está em formato *string*.

Este é um serviço necessário para um ambiente de agentes móveis, o qual terá basicamente objetos-agência e objetos-agente. Como os lugares de destino dos agentes estão representados pelas agências, então é adequado registrar as agências, como objetos Corba, nos serviços de nomes. O mesmo poderia aplicar-se aos agentes móveis, o que não se recomenda uma vez que agentes não necessariamente serão objetos Corba, e em virtude da sua volatilidade e da enorme quantidade de agentes que existiriam no ambiente. Não é adequado utilizar-se o serviço de nomes para salvar as referências dos agentes, porque se provocaria um tráfego adicional na rede, devido às atividades freqüentes de seu registro e exclusão. Será mais adequado que sejam registrados localmente na agência que os acolhe, sendo assim o acesso dos agentes feito através das agências e não diretamente. Logicamente, no escopo local da agência, pode-se usar o serviço de nomes para registrar os agentes, se estes forem objetos Corba.

3.1.4. MASIF

O MASIF [MASIF98] define uma arquitetura e especifica duas interfaces que permitem a construção de plataformas interoperáveis, para o suporte de agentes móveis de diferentes tecnologias de software (por exemplo, Aglet da IBM, MuBot da Crystaliz, AgentTcl da Darmouth College, MOA da Open Group, Odyssey da General Magic, etc.).

Na arquitetura distinguem-se três componentes: o sistemas de agentes, o agente e o *place*. O sistemas de agentes é a plataforma que permite a criação, execução, transferência e eliminação do agente.O *place* é um ambiente de execução para o agente (por exemplo, um *thread* ou um processo).O agente é um programa que tem a habilidade de se movimentar através dos sistemas de agentes. Em um sistema de agentes podem existir vários *places* e em cada *place* podem existir vários agentes. Os três componentes têm um nome e um endereço associado.

Tanto o agente como o sistema de agentes são objetos. O agente pode ser um objeto de qualquer tipo, ou seja, pode corresponder a objetos de diversas tecnologias. O sistema de

agentes é um objeto CORBA, mas que suporta só um tipo de agentes. Portanto, um agente poderá visitar só sistemas de agentes de seu tipo; por exemplo, um agente Aglet poderá visitar só sistemas de agentes construídos pela IBM para suportar agentes Aglets. Mas todos os sistemas de agentes, de qualquer tipo, podem comunicar entre si através das interfaces especificadas pelo MASIF.

O MASIF especifica as seguintes duas interfaces: *MAFFinder* e *MAFAgentSystem.*

3.1.4.1.O MAFFinder

A interface *MAFFinder* é o equivalente a um serviço de *Naming*, só que é capaz de gerenciar objetos de qualquer tipo, não somente objetos CORBA. Para poder gerenciar os agentes e os sistemas de agentes, propõe-se uma estrutura para o nome, definida pela estrutura *Name,* e um *string* para o endereço, definido pelo tipo *Location.*

A estrutura *Name*

Este serviço define uma estrutura para o nome dos agentes móveis e para o sistema de agentes (agência) da seguinte forma:

```
typedef sequence<octet> OctetString;
typedef OctetString Authority;
typedef OctetString Identity;
typedef short AgentSystemType;
struct Name {
        Authority           authority;
        Identity            identity;
        AgentSystemType     agent_system_type;
};
```

Onde:

authority: Representa o nome da pessoa ou organização dona do agente ou sistema de agentes. Este valor pode ser usado para autenticação.
identity: Diferencia um objeto de outro quando ambos têm o mesmo dono (*Authority*) e tipo.
agent_system_type: Indica o tipo de sistema ao qual pertence o agente ou *agent system.* Já tem alguns valores padronizados:

0 = Non Agent System
1 = Aglets
2 = MOA
3 = Agent Tcl

Esta estrutura de nomes cria um nome para os agentes e os sistemas de agentes que é globalmente único. Para ser compatível com a estrutura de nomes do Serviço de Nomes, a

especificação indica que cada um destes elementos será inserido como componente na estrutura anterior.

Como este sistema suporta a transferência de classes entre os sistemas de agentes, define-se um formato para os nomes das classes, para o qual não é garantido que seja globalmente único. O implementador é responsável para que esse nome seja único no domínio-origem do sistema de agentes (de onde é obtida a classe), tanto para as invocações *receive_agent()*, quanto para *create_agent()*.

Portanto, o nome dos objetos agente e sistema de agentes, não adere ao padrão do Serviço de Nomes. De fato, o MASIF define a interface *MAFFinder* como um serviço novo de nomes para o sistema de agentes.

A *string Location*

Para obter uma referência a um *MAFFinder*, o cliente pode usar o Serviço de Nomes ou o método *AgentSystem.get_MAFFinder()*. Posteriormente, pode obter a referência a um agente ou sistema de agentes, invocando o método *lookup_agent()* ou *lookup_agent_system()*, o qual retorna uma referência do tipo *Locations*, que está definida como uma seqüência de valores do tipo *Location*, que pela sua vez é do tipo *string*.

O *Location* pode ter um de dois formatos:

- Um URI [URI94] contendo um nome Corba. Neste caso, a *string* começa com o prefixo "CosNaming". Para resolver, deve-se pegar o valor *string* e convertê-lo para o formato *CosNaming.Name* e, em seguida, solicitar a um Serviço de Nomes a referência ao objeto. Exemplo:
  CosNaming:/agentes!unicamp/estudante!lca
  qual corresponde a um nome Corba com dois componentes, onde cada componente tem sua identidade(*id*) e tipo (*kind*). No exemplo:
  {"agentes", "unicamp"}, { "estudante","lca"}
- Um URL contendo um endereço Internet, com informação sobre a localização de um sistema de agentes. Neste caso o *string* começa com o prefixo "mafiiop" e o resto está definido por um formato contendo informação sobre o domínio e porta do hospedeiro, similar ao URL padrão, além de uma chave de objeto (uma série de octetos definida no documento do CORBA para a descrição de um IOR). Logo, a partir da informação contida no *string*, o cliente pode construir o IOR do sistema de agentes. Assim não é necessário consultar um Serviço de Nomes. Por exemplo, se um cliente invoca a operação *MAFFinder.lookup_agent()*, utilizando o nome de um agente, receberá como resposta um dado do tipo *Location*, contendo indiretamente o endereço de um sistema de agentes, onde o agente está residindo.

O formato URL definido pelo MASIF, não é igual ao formato URL que o Serviço de Nomes (interoperáveis) está definindo. Para o nome do *place* define-se um formato *string* simples que só tem significado dentro do domínio do sistema de agentes que acolhe o

place. Com relação à sua localização, o endereço do *place* está associado ao endereço do *agent system* de forma similar ao agente.

É importante salientar que, no MASIF, os agentes podem ou não ser objetos Corba, mas os sistemas de agentes deverão sê-los sempre. Por outro lado, o MASIF utiliza a interface *MAFFinder* como serviço de nomes alternativo ao Serviço de Nome da OMG para gerenciar os nomes dos agentes, sistemas de agentes e *places*.

O MASIF define o conceito de **região**, o qual corresponde a um conjunto de sistemas de agentes com o mesmo dono (*authority*) mas que podem ser de tipos diferentes. Por simplicidade, uma região pode ser gerenciada por um *MAFFinder*. Mas nada impede que um *MAFFinder* seja compartilhado por mais de uma região.

Quando o objeto é criado e registrado com o *MAFFinder*, o objeto de referência é convertido em *string* e armazenado como componente do URL no parâmetro *agentsystem*. Logo, este esquema permite armazenar e gerenciar nomes e endereços de objetos de diversos fabricantes (não necessariamente objetos CORBA), marcando diferenças em relação ao serviço *Naming* do CORBA.

Além disso, define-se a estrutura *AgentSystemInfo* para o sistema de agentes e a estrutura *AgentProfile* para o agente. É importante salientar que, em ambas as estruturas, há parâmetros que são padronizados pelo MASIF, os quais podem ser entendidos por clientes de qualquer tipo. A partir deles, o cliente determina se o objeto que está manipulando é de seu tipo e, em caso positivo, o cliente poderá entender o resto dos parâmetros. Por exemplo, no caso *AgentProfile*, os parâmetros: *language_id*, *agent_system_type* e *serialization*, são padronizados pela OMG, e os restantes dependerão do fabricante do objeto. Neste ponto, é importante salientar que o MASIF tem, como trabalho futuro, tentar padronizar os nomes das propriedades dos objetos definidos pelo parâmetro *properties*: atualmente as propriedades são específicas da aplicação.

Com relação ao nome e endereço do *place*, o nome é do tipo *string*, isto significa que não possui nenhuma estrutura. O endereço é do tipo *Location*, o qual como já se falou, tem um formato URL. O endereço do *place* está definido por dois dados: um é o valor do parâmetro *agentsystem*, que deve corresponder a um endereço (referência do objeto) de um objeto do tipo sistema de agentes, onde está alojado o *place*; o outro é o valor indicado pelo parâmetro *place*, que corresponderia a um *string* contendo um *path*. Logo, convertendo o parâmetro *agentsystem* para referência do objeto (*stringified*), é possível acessar o sistema de agentes onde está o place. Em seguida, pegando o valor do parâmetro *place*, deveria ser possível acessar o *place*: note-se que tudo é especifico do tipo ao qual pertence o sistema de agentes, ou seja, tem de ser definido pelo fabricante.

Portanto o *Location* não pode ter um formato *CosNaming*, porque o *place* não é um objeto no sentido tradicional, ou seja, não existe uma referência de objeto que o identifique. Uma

vez que o **place** não tem uma estrutura definida, só um nome de *place* poderá existir na Base de Dados, gerenciada por um *MAFFinder*.

3.1.4.2.O MAFAgentSystem

Esta interface permite criar, transferir, suspender, reiniciar e eliminar (*terminate*) um agente, além de outras ações que estão mais relacionadas com a informação que um sistema de agentes possui.

Para transferir um agente, a interface fornece duas operações: *receive_agent()* e *fetch_class()*.

Quando um agente toma a decisão de migrar para um outro local, comunica-o ao sistema de agentes que o suporta, de maneira **não definida na especificação** (por exemplo, através de uma operação "*go()*"). Posteriormente o sistema de agentes invoca a operação *get_agent_system_info()* para determinar se o destino suporta o tipo do agente. Se o tipo não é suportado, invocará a operação *find_nearby_agent_system_of_profile()* para achar o sistema de agentes adequado mais perto do destino. Posteriormente, invocará a operação *receive_agent()* do sistema de agentes destino achado. Com essa operação, será transferido o código básico para executar o agente. Se é necessário transferir mais código, o sistema de agentes destino poderá invocar a operação *fetch_class()*, do sistema de agentes origem, para pegar o resto do código. Cada sistema definirá qual algoritmo usar ou quando transferir o resto do código. Por exemplo, pode-se proceder a transferir o resto do código quando o agente está sendo executado (a parte básica) ou à medida que for necessário: a especificação não se pronuncia a esse respeito.

A especificação define três estados para um agente:

CfMAFRunning: O agente está em execução.

CfMAFSupended: O agente não está em execução.

CfMAFTerminated: O agente finalizou sua execução.

É importante salientar que o sistema de agentes-destino, poderá executar o agente em um outro sistema de agentes, de forma transparente para quem solicitou a transferência. Desta forma pode-se ter um conjunto de sistemas de agentes gerenciados por um deles, e que permanece público para o resto do sistema, oferecendo-se assim maior disponibilidade para receber agentes de vários tipos.

As operações *resume_agent()* e *suspend_agent()* da forma que são descritas pela especificação e da forma como são utilizadas no exemplo do cenário da aplicação, sugerem

que, suspender e reiniciar um agente, significa suspendê-lo e reiniciá-lo no mesmo lugar, para realizar-se alguma modificação (por exemplo, no itinerário do agente, mudar as condições de busca, etc.), mas não com o fim de proceder a mover o agente para um outro local e reiniciá-lo lá.

A questão de serialização do agente fica aberta, como também a forma de transferência dessa informação para o destino: cada sistema definirá seu próprio esquema. Em todo caso a especificação define alguns tipos de serialização, até agora, só o correspondente a Java. Estranhamente não se inclui o serviço CORBA de externalização.

3.1.4.3.Migração e IOR

Quando um agente migra, sua referência de objeto muda, o que significa que a lista de agentes com seus endereços terá que ser modificada. Esta atividade será executada muito freqüentemente devido à mobilidade dos agentes. Além disso, os clientes que estavam se comunicando com o agente terão que atualizar, de alguma forma, a referência que tenham sobre o agente. Com relação a este último problema, o MASIF opta por atualizar o serviço de Nomes com o novo IOR. Contudo é necessária a atenção do cliente. Por exemplo, se o agente migra, o cliente recebe uma mensagem de exceção indicando *invalid object reference*. Neste caso, o programa cliente deverá manipular esta excepção acessando o serviço de nomes (o *MAFFinder*) para receber o novo endereço do agente, e proceder de novo ao estabelecimento da ligação.

Esta solução é totalmente transparente para a aplicação, ou seja, executada pelos ORBs, utilizando as mensagens GIOP adequadas e, portanto, consumindo recursos adicionais da plataforma. Contudo a implementação desse esquema não é obrigatória para os fabricantes de ORBs.

É interessante aqui fazer uma comparação com os mecanismos de migração de objetos usados por um dos ORBs do mercado, o Visibroker. Quando é necessário mover um objeto de um local para outro, primeiro deve-se instanciar o objeto no local de destino, ficando registrado esse objeto no Repositório de Implementação. Em seguida procede-se à eliminação do objeto, no lugar de origem. O *"osagent"*, programa encarregado de monitorar as referências de objeto, vai perceber esta situação e imediatamente procurará um objeto com o mesmo nome e formato das interfaces. Quando encontrar o objeto procederá à atualização internamente das referências de objeto, de forma transparente para a aplicação. Estes procedimentos não são definidos no padrão CORBA .

3.1.4.4.Uso dos CORBAservices

MASIF não utiliza o serviço de Nomes e define o *MAFFinder* para implementar as funções de gerenciar o agente.

Do serviço *LifeCycle* fala-se que nada impede usá-lo nos objetos do tipo *MAFFinder* e *MAFAgentSystem*, porque eles são objetos CORBA. No caso do agente, o seu uso só será possível se o agente for um objeto CORBA. De qualquer forma, a sua utilização só é a responsabilidade do implementador do sistema. O mesmo acontece com o serviço de Externalização.

Também não se exclui o uso do serviço de Segurança, para aplicá-lo à transmissão do agente, indicando-se, para tal, algumas sugestões. Além de autenticar o agente, é necessário que os sistemas de agentes participantes, na transferência do agente, também se autentiquem mutuamente. Por outro lado, indica-se que a especificação atual do serviço de segurança não aborda a questão da autenticação entre vários domínios, que é o caso do agente quando viaja entre domínios diferentes (*multi-hop travel*).

3.1.4.5.Comentários sobre o MASIF

- A especificação padroniza a interoperação entre as plataformas de diversos fabricantes de tecnologias de agentes. Para isso, define uma interface destinada a implementar um serviço de nomes para agentes de qualquer tipo e uma interface que controla o comportamento da plataforma para suportar agentes móveis.
- A especificação é relativamente genérica, não impondo mecanismos para salvar e transmitir o estado junto com o código, o que poderia apresentar problemas de portabilidade de uma plataforma para outra (*marshalling*). O uso de outros serviços CORBA também não é prescrito
- O transporte do código do agente é feito através de parâmetros do tipo *string* de octetos das operações já descritas, os quais não estão sujeitos ao processo de *marshalling* e sem definir detalhes sobre o formato desses parâmetros, permitindo aos fabricantes definir seus próprios esquemas para transmitir um objeto ou um grupo de objetos ou um conjunto de objetos, mais outros tipos de arquivos. Isso significa que o transporte é feito usando IIOP, não se especificando outras alternativas para transmitir o agente, por exemplo, com a escolha de uma via mais rápida de transporte.
- A negociação do agente só implica em saber se o sistema de agentes-destino é capaz de suportar o tipo de agente que se quer enviar; após isso, o agente é enviado sem mais considerações.
- Pelo fato de se usar um serviço de nomes diferente do já especificado pela OMG, com o fim de gerenciar objetos pertencentes a diversos modelos (não somente CORBA), faz-se necessário um passo adicional para obter a referência do objeto. Se considerarmos que isto vai acontecer freqüentemente, pelas características de movimentação de um agente, pode-se ter um impacto importante no desempenho da aplicação baseada em agentes móveis. O uso do serviço de nomes da OMG implicaria necessariamente na construção dos agentes como objetos CORBA, condição comercialmente não aconselhável. Atualmente está em estudo a especificação de um serviço de nomes interoperáveis que, entre outras coisas, terá como objetivo modificar a estrutura do nome dos objetos, só dando suporte, aparentemente, ao registro de objetos CORBA.

- Como o sistema de agentes é definido pela estrutura *AgentSystemInfo*, onde o parâmetro *agent_system_type* define o tipo ao qual pertence o sistema de agentes, cada um deles terá só um tipo. É evidente que cada sistema de agentes é implementado por um fabricante de tecnologias de agentes, unicamente para seus próprios agentes. Será responsabilidade do implementador, organizar diferentes tipos de sistema de agentes sob uma autoridade (região), para receber diferentes tipos de agentes que serão repassados ao sistema de agentes específico dentro da região. Não se define nenhuma estrutura geral para o agente: de fato ele pode ser de qualquer tipo, não necessariamente CORBA.
- Deixa-se para o futuro, a definição dos nomes das propriedades de um agente.
- Sobre os protocolos de transporte, este padrão especifica que a transferência do agente e de suas classes pode ser feita usando as operações *receive_agent()* e *fetch_class()* da interface *MAFAgentSystem*. Isto significa que a transferência é feita usando IIOP, mas nada impede que outros protocolos sejam utilizados.

Acredita-se que este seja o primeiro passo na padronização de um Serviço de Facilidades para agentes móveis e que o amadurecimento desta tecnologia permita avanços futuros nesse processo. Atualmente as plataformas de agentes móveis, como Aglet, Grasshopper, MOA, D'Agents (ex Agent Tcl), estão anunciando que seus produtos são compatíveis com o padrão do MASIF.

3.1.5. GIOP e movimentação de objetos sob CORBA

A partir da versão 2.2 da especificação do CORBA, considera-se a movimentação de objetos a nível de GIOP (*General Inter-ORB Protocol*), para que esta questão fique transparente para o cliente. A seguir, vão ser descritos os mecanismo que CORBA especifica para suportar a movimentação dos objetos CORBA, chamados de migração de objetos [Henning98]. Na realidade trata-se do suporte ao translado de um objeto CORBA de um local para outro, sem considerar seu estado. Sem dúvida, este mecanismo é útil para implementar ORBs que suportem a migração de objetos CORBA, ou seja a movimentação de objetos considerando seu estado.

Dentro das sete mensagens GIOP especificadas por CORBA, as seguintes têm a ver com a movimentação dos objetos:

- ***LocateRequest***: Esta mensagem pode ser enviada pelo cliente para o servidor, com o fim de obter a seguinte informação com relação à referência de um objeto:
  - Se a referência de um objeto é válida,
  - se o servidor atual é ou não capaz de receber diretamente os requisições para sua referência,

- para saber qual é o novo endereço da referência do objeto, ao qual devem-se enviar os requisições.

- ***LocateReply***: Esta é a mensagem que o servidor manda ao cliente como resposta a uma mensagem *LocateRequest.*

Estas mensagens permitem implementar ORBs para controlar a referência dos objetos que se movimentam de forma transparente para a aplicação. Isto significa que é útil para a comunicação entre objetos que se movimentam na rede, como é o caso dos agentes móveis. Na especificação, considera-se como suporte à migração de objetos, mas que de acordo com a nossa definição de migração, nada tem a ver com o controle da referência dos objetos móveis.

Quando o lado cliente inicia um requisição, seu ORB local consulta o servidor, emitindo um *LocateRequest*, se o objeto ainda estiver no lugar. Se não está, então a mensagem de resposta do *LocateReplay* contém o novo endereço do objeto. Isto significa que o servidor terá controle desta situação e sempre que um objeto se movimentar, deverá deixar indicado o novo endereço, ou seja, o ORB do lado servidor deverá administrar uma tabela de endereços para os esses objetos e atender às consultas dos clientes.

Este mecanismo raramente é implementado nos atuais ORBs comerciais, talvez por problemas de desempenho e tamanho dos produtos ou por falta de aplicação.

3.1.6. Adaptador de Objetos Portáveis (POA: *Portable Object Adapter*)

Como se pode observar do modelo de referência do CORBA [POA99], o Adaptador de Objetos é um objeto que tem como função mediar diretamente entre o ORB e o objeto de implementação, e indiretamente através do *Skeleton* e do DSI (*Dynamic Skeleton Interface*), cooperando com o ORB, para que a requisiçao do cliente seja atendida pela implementação.

Até antes da versão 2.2 da especificação do CORBA, este mediador era parcialmente especificado, deixando para o fornecedor do ORB a decisão de definir as interfaces entre o adaptador de objetos e o ORB principalmente, além de outros aspectos como gerenciamento dos objetos dentro do programa servidor (*server*). Como resultado, as implementações dos objetos CORBA não são portáteis entre os ORBs de fabricantes diferentes.

A partir da versão 2.2 do CORBA, especificou-se um adaptador de objetos que permite a portabilidade das implementações dos objetos CORBA, nomeado como Adaptador de Objetos Portáveis (POA: *Portable Object Adapter*).

Basicamente, a especificação padroniza a forma de gerenciar os objetos utilizando objetos POAs, os quais associam-se a objetos computacionais reais, nomeados *servants*, que

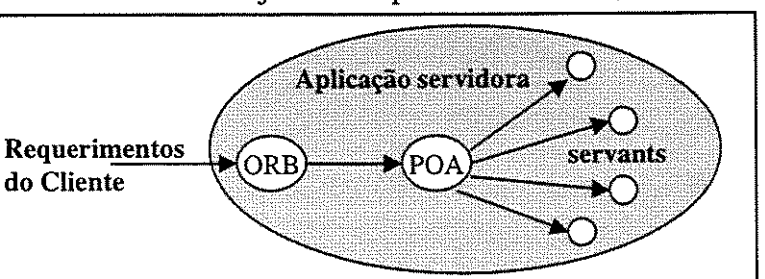

Figura 3.4: Gerenciamento dos *servants* através do POA

representam uma implementação do objeto CORBA residente no servidor, como se observa na figura 3.4. Os objetos *servants* são gerenciados pelo POA, baseados em diretrizes definidas pelo implementador do objeto, dentro de um conjunto de regras definidas pela especificação, nomeadas de políticas. Não se padroniza a forma de instanciar um objeto dentro do servidor, aspecto próprio da linguagem de programação utilizada.

A interface POA permite, ao implementador dos objetos CORBA, um maior controle na criação e execução dos objetos. Por exemplo, entre outros, pode definir se que o objeto seja executado num ambiente *multithread*, que seja persistente, que seja ativado pelo servidor ou de forma implícita, ou que seja usado um mesmo *servant* para implementar diferentes objetos, tendo cada *servant* uma única identificação. A forma de despacho pelo ORB, para um POA, da requisição de um cliente, criado dentro do programa servidor, é responsabilidade do fabricante do ORB, ou seja, a interface entre o ORB e o adaptador de objetos não está padronizada [Vinoski98].

O fabricante do ORB proverá ao implementador da aplicação, no lado servidor, um adaptador de objetos com interfaces padronizadas pelo POA, característica que anteriormente não existia nos ORBs. O POA oferece a possibilidade de definir características de execução dos objetos, dentro de um conjunto finito de possibilidades, consideradas como as mais comuns, deixando ao fabricante do ORB a liberdade de implementar algumas ou todas essas possibilidades. Outras alternativas poderão ser implementadas, mas assume-se que dificilmente virão a ser encontradas em outros ORBs.

POAs não serão portáteis entre ORBs de diferentes fabricantes que tenham aderido às especificações POA, mas sim os objetos de implementação.

3.1.6.1.Funcionamento do POA

Como é usual, o programa servidor é o encarregado de instanciar os objetos. Primeiro, deverá criar um objeto ORB, posteriormente através dele gerar um objeto POA, que será a

raiz de outros POAs que possam vir a ser instanciados no servidor: este objeto é denominado de *rootPOA*, e já terá suas políticas definidas "a priori". Se o implementador achar que o POA-raiz é suficiente para os objetos de implementação, continuará instanciando objetos de implementação CORBA: estas instâncias serão os *servants*. A sua geração será feita com as ferramentas fornecidas pelo ambiente da linguagem de programação.

O objeto instanciado será registrado no POA, para que fique associado a uma referência de objeto CORBA, ficando publicado e pronto para que seja acessado pelos clientes. Posteriormente o programa servidor ativará o objeto ORB para atender as requisições

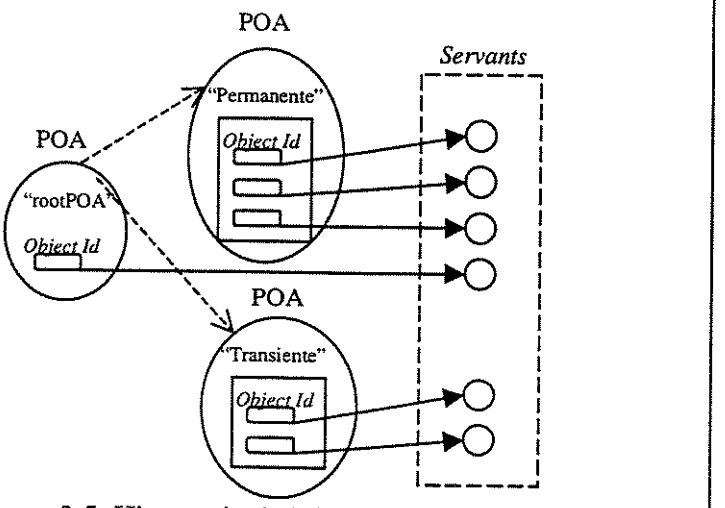

Figura 3.5: Hierarquia de POAs com diferentes políticas

remotas. Se o implementador achar que as políticas do *rootPOA* não são adequadas a suas necessidades, então criará um objeto, definido pela interface *Policy*, com as políticas que desejar, associando o POA criado ao *rootPOA*.

Cada POA (Figura 3.5) gera uma identificação para cada objeto que gerencia, conhecida como *Object Id* : essa identificação tem validade dentro do POA. Fora dele, acrescenta-se o nome do POA, para que o ORB possa identificar o POA ao qual pertence o objeto requisitado. No caso de um POA ser filho de outro, o nome deve refletir a hierarquia ao qual pertence. Para gerenciar estas identificações, são especificadas interfaces que representam objetos ativadores de adaptadores (*AdapterActivator*) e gerenciadores de *servants* (*ServantManager*): seu uso dependerá das políticas escolhidas.

Tanto os ORBs, POAs , objetos ativadores de adaptadores e gerenciadores de *servants*, são objetos que interagem localmente (pseudo-objetos CORBA), no processo do programa servidor, para cooperar na entrega, para o *servant*, da requisição solicitada por um cliente. Somente os *servants* poderão receber invocações remotas, representando assim os objetos CORBA.

Na Tabela 1 apresenta-se um resumo das diferentes políticas suportadas pelo POA.

Thread	(*) ***ORB_CTRL_MODEL***: O ORB escolhe o modelo de *multithread* adequado.
	SINGLE_THREAD_MODEL: Garante que todos as requisições, para os objetos no POA correspondente, serão executadas em um único *thread*.
LifeSpan	(*) ***TRANSIENT***: Os objetos CORBA criados pelo POA não sobrevivem ao término do processo no qual foram criados.
	PERSISTENT: Nesta política sim sobrevivem.
ObjectId Uniqueness	(*) ***UNIQUE_ID***: Os *servants* criados pelo POA suportam um ObjectId.
	MULTIPLE_ID: Os *servants* criados pelo POA suportam um ou mais *ObjectId*.
Id Assignment	(*) ***SYSTEM_ID***: Os *ObjectId* são consignados pelo POA. Se o POA também é *PERSISTENT*, então os *ObjectId* devem ser únicos através de todas as instanciações do objeto feito pelo POA.
	USER_ID: Em aqueles objetos criados pelo POA com esta política, seus *ObjectId* são consignados somente pela aplicação.
Servant Retention	(*) ***RETAIN***: O POA reterá os *servants* ativos mantendo seus *ObjectId* na tabela *Active Object Map*.
	NON_RETAIN: Os *servant* não são retidos pelo POA. Neste caso, deve-se trabalhar conjuntamente com a política *Request Processing*, definida como: *USE_DEFAULT_SERVANT* ou *USE_SERVANT_MANAGER*.
Request Processing	(*) ***USE_ACTIVE_OBJECT_MAP_ONLY***: Se o *ObjectId*, de um *servant* requisitado, não está na tabela *Active Object Map*, então será gerada a excepção *OBJECT_NOT_EXIST*. Neste caso, deve-se trabalhar juntamente com a política *Servant Retention* definida como *RETAIN*.
	USE_DEFAULT_SERVANT: No caso que o *ObjectId*, do *servant* requisitado, não estiver na tabela *Active Object Map*, quando a política *RETAIN* está sendo usada, ou se a política *NON_RETAIN* está sendo usada, em ambos casos a requisição será despachada a um *servant* implícito (*default servant*) associado ao POA, o qual deverá existir previamente.
	USE_SERVANT_MANAGER: Do contrário, o requisição será enviado ao objeto *servant manager*, quem localizará o *servant* ou gerará uma excepção.
Implicit Activation	(*) ***NO_ IMPLICIT_ACTIVATION***: O POA não suportará a ativação implícita dos *servants*.
	(*) ***IMPLICIT_ACTIVATION***: O POA suportará a ativação implícita dos *servants*.
(*): São os valores do "rootPOA" e de qualquer POA quando a política não seja definida	

expressamente pelo implementador. (*)Para que seja utilizada a ativação implícita dos *servants*, devem-se definir as políticas como *IMPLICIT_ACTIVATION, SYSTEM_ID* e *RETAIN*
Tabela 1: Políticas suportadas pelo POA.

3.1.6.2.O POA na mobilidade dos agentes

A padronização de um adaptador de objetos, que seja portável para diferentes plataformas, é muito importante para os agentes móveis, porque a portabilidade é um fator importante no sucesso da mobilidade dos agentes, se estes estão baseados em CORBA. Este padrão permite que a ativação dos agentes móveis seja uniforme nas diferentes plataformas (com ORBs de diferentes fabricantes) que o agente visite. Note-se, contudo, que o transporte do código fonte do agente e a sua compilação no destino continuam como antes necessários.

3.1.7. Componentes CORBA

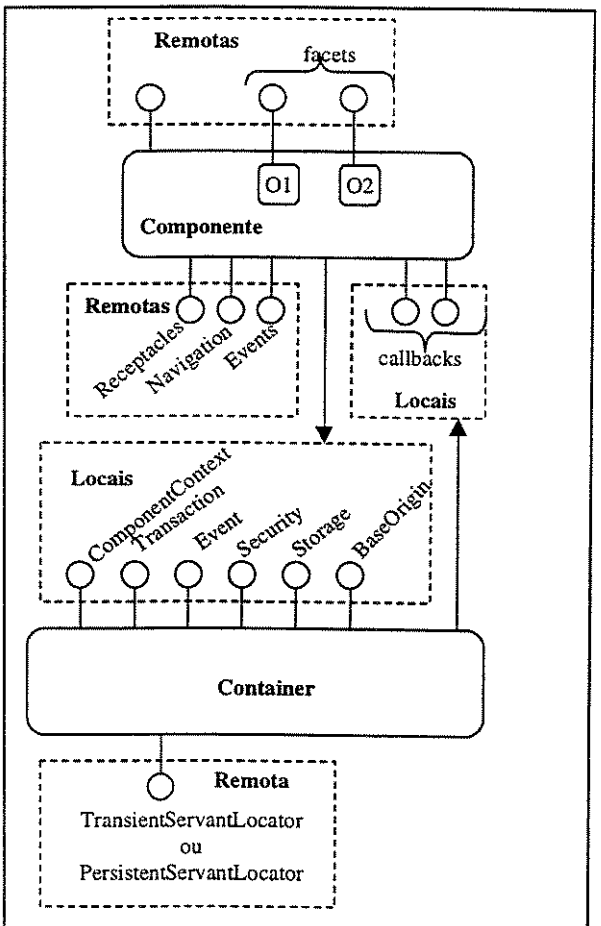

Figura 3.6: Modelo do Componente CORBA

Descreve-se, nesta seção, o modelo de componentes que a OMG está definindo [Compon99]. O objetivo desta descrição é dar uma visão geral sobre as diversas partes deste modelo e como elas interagem entre si e com o resto da arquitetura CORBA, especialmente com o POA (*Portable Object Adapter*).

O modelo de componentes define as interfaces de um componente CORBA, a forma do ambiente de execução (*Container*) para esses componentes e a forma de introduzir, instalar, configurar e ativar componentes (*deployment*).

Na figura 3.6, apresentam-se as principais partes de um sistema baseado em componentes, o **objeto componente** e o **objeto *container***, encarregado de controlar a execução do componente, além das interfaces mais relevantes para uma descrição do funcionamento global do modelo de componentes.

Como se mostra na figura, há dois tipos de interfaces, locais e remotas. As primeiras são interfaces que podem ser só acessadas localmente, similares aos pseudo-objetos

CORBA, como os ORB e BOA definidos nas versões anteriores da especificação CORBA. A partir da versão 3.0 vai-se introduzir o termo *local*, para especificar uma interface de acesso unicamente local e que não estará afeta ao *marshalling*. As interfaces indicadas na figura como remotas são interfaces tradicionais, ou seja, elas podem ser acessadas partir de qualquer lugar remoto ao hospedeiro que contém os objetos referenciados. Isto significa uma mudança nos compiladores IDL, para que suportem este novo tipo de interface, além de outras mudanças documentadas no capítulo 12 da especificação [Compon99].

É importante salientar-se que haverá um objeto componente, com interfaces de acesso remoto e local, o qual vai interagir localmente com seu *container* e remotamente com outros objetos Corba. Como já se falou, ambos os tipos de interfaces poderão ser especificados numa versão atualizada do IDL.

3.1.7.1.O componente

O componente tem uma interface *Navigation*, através da qual se poderá acessar as outras interfaces internas. Cada objeto que seja parte do componente apresenta suas interfaces, definidas como *facets*. Para que um cliente possa acessar estes objetos, primeiro deve ter uma referência do componente, para então acessar a interface *Navigation* e receber a referência do objeto desejado.

As interfaces *Receptacles* e *Events* permitem ligar o componente com outros componentes e objetos, e assim desenvolver aplicações por montagem (*assembly*) desses componentes. No caso da interface *Receptacles*, ela permite receber referências de objetos externos, que os objetos do componente utilizem para seu funcionamento. No momento da configuração, as referências dos objetos que o componente utiliza, as quais estão registradas nos arquivos para instalação e configuração (*deployment*) do componente, serão repassados através desta interface. A interface *Events* permite que o componente possa ligar-se num canal de eventos, a partir do qual poderá receber ou enviar eventos: assim o componente poderá comunicar-se com outros via eventos. Além disso, esta interface permite o gerenciamento das interfaces de evento do componente que participa da comunicação.

As interfaces *callbacks* permitem ao *Container* controlar os componentes. Através delas, o *Container* pode ativá-los e/ou desativá-los (*passivate*). Na realidade, para cada componente, define-se uma interface *callback*, onde seu tipo depende da categoria à qual pertence o componente e que pode ser uma das quatro categorias seguintes: *Service*, *Session*, *Process* e *Entity*. Quando o objeto componente é criado, o *Container* fica com a referência desse objeto, através do qual poderá acessar a interface *callback*, para gerenciá-lo.

Da mesma forma que o serviço *LifeCyle* gerencia o ciclo de vida de um objeto, no caso dos componentes especificou-se um conjunto de interfaces para gerenciar o ciclo de vida dos mesmos. Para a fabricação (*Factory*) dos componentes utilizam-se as interfaces do tipo *Home* (*HomeBase, KeylessHomeBas e HomeFinder*) e para sua configuração utilizam-se as interfaces: *Configurator, ConfigValue, StandardConfigurator e HomeConfiguration.* Por simplicidade, na figura 3.6, não foram incorporados objetos fabricadores nem de

configuração, mas eles devem existir previamente à instanciação dos objetos componentes. Para cada categoria de componente existirá um *Home* ou fabricante de componentes.

3.1.7.2.Relacionamento entre as interfaces do componente

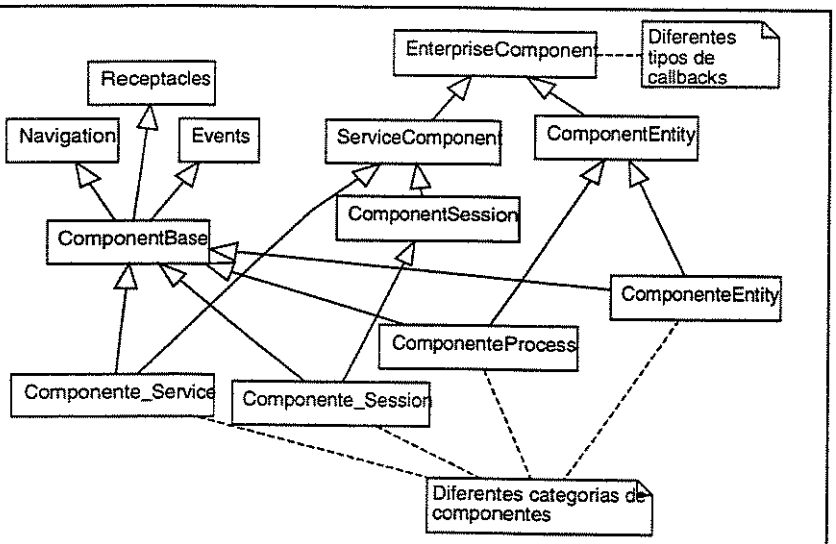

Figura 3.7: Interfaces das diferentes categorias de componentes: *Session, Service, Process* e *Entity*

Na figura 3.7 apresenta-se o relacionamento das diversas interfaces que compõem um componente. Como se observa, qualquer que seja a categoria do componente, ele deve herdar a interface *ComponentBase*, através da qual incorporam-se as interfaces *Receptacle*, *Navigation* e *Events*. Dependendo da categoria à qual pertence o componente, ele deve herdar um tipo de interface *callback*, por exemplo: *ServiceComponent*, para a categoria *Service*, ou *SessionComponent*, para a categoria *Session*, ou *EntityComponent* para as categorias *Process* e *Entity*.

3.1.7.3.O Container

O *Container* é a entidade encarregada de fornecer um ambiente de execução padronizado para os componentes. Desta forma, os componentes desenvolvidos por qualquer fabricante poderão ser executados em ambientes suportados por outros fabricantes ou vendedores de ORBs. Por estar muito ligado à execução dos objetos, ele foi construído a partir de uma especialização do POA (*Portable Object Adapter*). Ou seja, o *Container* é um POA ao qual foram incorporadas outras interfaces (especialização) para que possa gerenciar adequadamente os componentes que receba. Na figura 3.6, apresentam-se as mais relevantes, para explicar o funcionamento do modelo. O componente não pode ter acesso direto aos serviços oferecidos no local, fazendo-o através das interfaces locais do *container.*

Toda vez que um componente é criado, a referência à interface local *ComponentContext* do *container* , que fornece seu ambiente de execução, é repassada ao componente. Desta forma poderá ter acesso aos seguintes serviços básicos oferecidos pelo *container*: transações, eventos, segurança e persistência.

Através da interface local *BaseOrigin*, o componente pode solicitar ser desativado (*passivate*). Esta interface pode ser especializada para incorporar outras necessidades dos componentes, que sejam adequadas para um tipo de aplicação.

Além das interfaces apresentadas na Figura 3.6, o *container* oferece interfaces locais que permitem aos componentes registrar seus fabricantes (*Home*), para que eles possam ser encontrados pelos *HomeFinder*. Estas interfaces são: *HomeRegistration* e *RemoteHomeRegistration*.

A partir da especificação, podem se construir dois tipos de *containers*: *Session* e *Entity*. Os primeiros suportam os componentes das categorias *Service* e *Session*. Os *containers* do tipo *Entity* suportam os componentes das categorias *Process* e *Entity*. Isto significa que conjuntos de interfaces diferentes devem ser acrescentadas às já indicadas. Nas Figuras 3.8 e 3.9 apresentam-se as interfaces e suas relações para os dois tipos de *containers*.

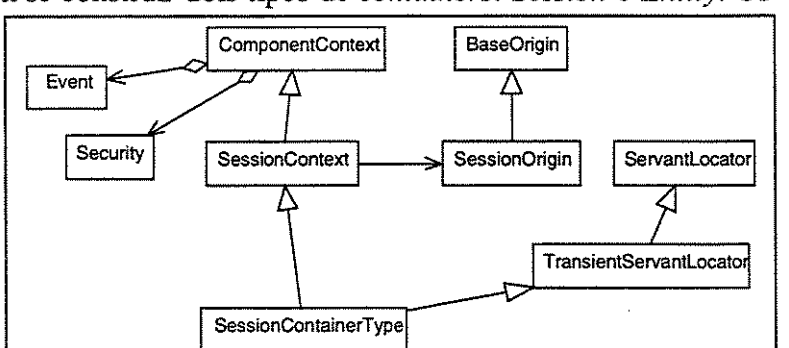

Figura 3.8: API para o *Container* do tipo *Session*

Por exemplo, para construir um *container* do tipo *Session*, que é nomeado como *SessionContainerType*, devem-se acrescentar as interfaces *SessionContext* e *SessionOrigin*, para manipular o componente como uma unidade só. A ausência da interface *Transaction* significa que este tipo de *container* não suporta esta facilidade, pelo menos na atual versão proposta. Também, não suporta nenhuma forma de armazenamento persistente. Além das anteriores, acrescenta-se a interface remota *TransientServantLocator*, especificada no módulo *Container*, a qual permite gerenciar o *container* para manipular objetos transientes.

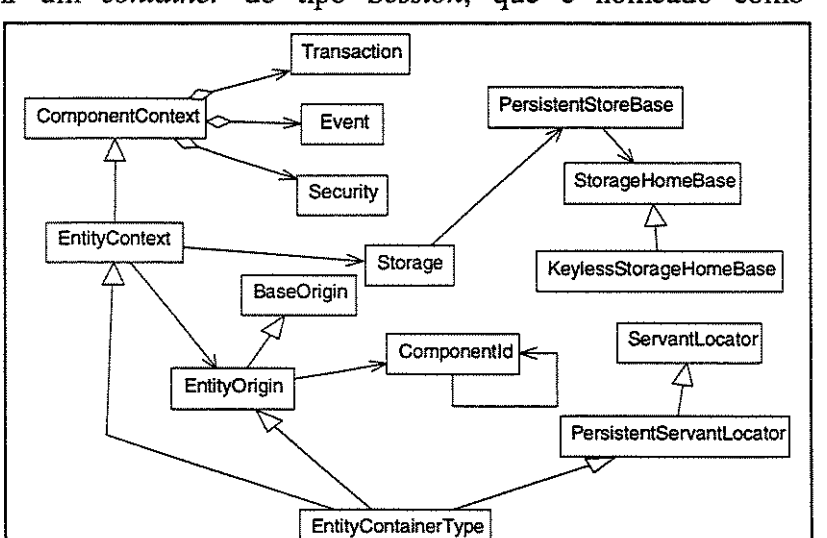

Figura 3.9: API para o *Container* do tipo *Entity*

Para implementar um *container* do tipo *Entity*, que é nomeado como *EntityContainerType*, devem-se acrescentar as interfaces: *EntityContext*, a qual serve para obter referência às interfaces encarregadas da persistência (*Storage*, *PersistentStoreBase*, *StorageHomeBase*,

KeylessStorageHomeBase), e as interfaces *EntityOrigin* e *ComponentId,* para a manipulação do componente. Este tipo de *container* oferece facilidades para usar transações, através da interface *Transaction,* e para implementar componentes incluindo objetos com estado, através das interfaces, já mencionadas, encarregadas da persistência e também para manipular objetos permanentes, através da interface remota *PersistentServantLocator.*

Todas estas interfaces locais, menos *TransientServantLocator* e *PersistentServantLocator,* encontram-se especificadas dentro do módulo nomeado *Server.* Entretanto, as interfaces remotas que permitem controlar o *container* como um POA especializado, estão especificadas no módulo nomeado *Container.* Os autores do *roposta* apresentam dois tipos de interfaces, nomeadas: *TransientServantLocator* e *PersistentServantLocator,* as quais herdam a interface *ServantLocator* do POA. Elas são apresentadas não como interfaces mandatórias mas sim como um guia para especificar *containers* que cumpram com os requisitos básicos descritos pelas interfaces já indicadas.

3.1.7.4.*Facets* e segmentos

Os *facets* contidos em um componente, podem ser agrupados no que o autores definem como **segmentos**. Só os *facets,* pertencentes a um segmento que contenha o *facet* invocado, serão ativados e não todos os *facets* do componente. É responsabilidade do implementador do componente definir estes agrupamentos. Para o gerenciamento destes segmentos, fornecem-se as interfaces locais *ComponentId* e *EntityOrigin* pertencentes ao *container.*

Os *containers* do tipo *Session* não fazem uso dos segmentos e manipulam componentes de um segmento. Só os *containers* do tipo *Entity* manipulam componentes com múltiplos segmentos.

É importante salientar que todos estes detalhes serão transparentes para os clientes do componente, ou seja, o cliente preocupa-se em obter uma referência do componente, a partir do qual terá acesso à interface *Navigation,* onde poderá obter a referência do objeto desejado (*facet*). Será responsabilidade do implementador do componente agrupar os objetos em segmentos para sua ativação.

3.1.7.5.Montagem dos componentes (*deployment*)

Um componente pode ser implementado em diferentes linguagens, para diferentes plataformas. Todas estas implementações podem ser reunidas e empacotadas para que o usuário da aplicação escolha a implementação mais adequada para seu sistema. Além disso os componentes podem ser agrupados, através das interfaces *receptacles* e *events* do componente.

Para permitir a manipulação dos componentes, como a sua montagem, através de ferramentas computacionais, foram especificadas as seguintes interfaces para gerenciar a instalação, configuração, ativação e desativação dos componentes, especificadas no módulo

nomeado *Deployment: Installation, Assembly Factory, Assembly, ServerActivator,*

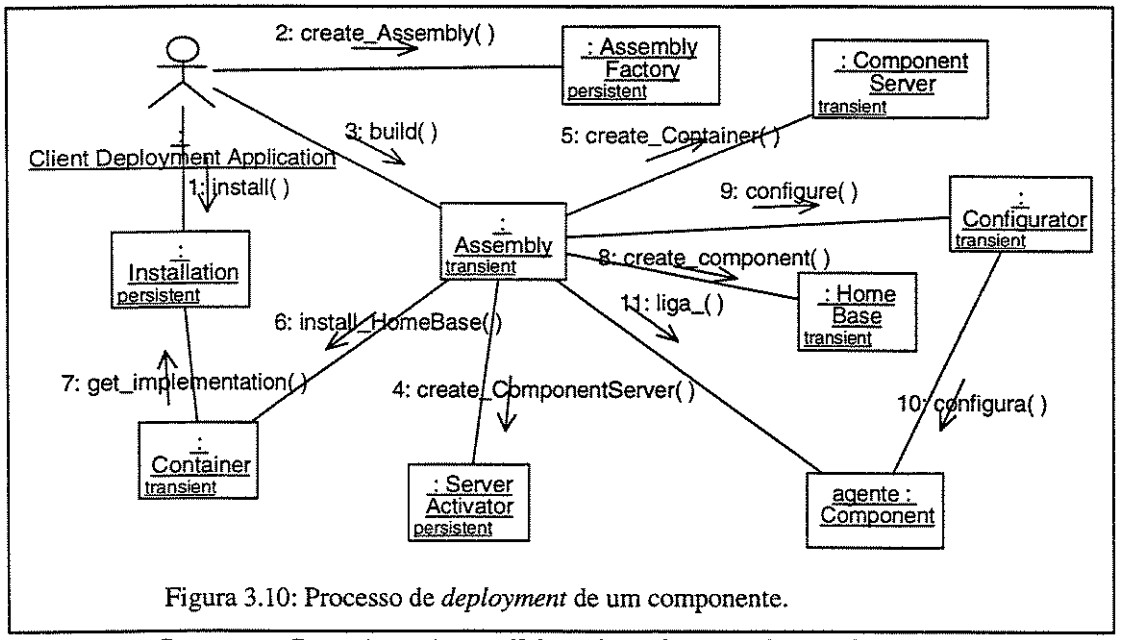

Figura 3.10: Processo de *deployment* de um componente.

ComponnentServer e Container (esta última interface pode confundir-se com o módulo *Container)*. Na figura 3.10 apresenta-se um diagrama de colaboração para o processo de *deployment* de um componente. Durante este processo, os objetos que participam na montagem trabalham com vários tipos de arquivos descritores de componentes para extrair a informação necessária.

O documento especifica a ordem de empacotamento de implementações de componentes, junto com os diferente tipos de arquivos descritores. Desta forma, o processo da montagem do componente está ligado com o empacotamento dos componentes.

3.1.7.6.Arquivos descritores e pacotes

Especificam-se três tipos de arquivos descritores, os quais são escritos usando o formato XML. Eles descrevem a forma de empacotar o componente.

Descritor tipo ccd

Primeiramente, quando o desenvolvedor de componentes constroi uma implementação em uma linguagem e para uma plataforma específica, criará (com ajuda do compilador da linguagem CIDL descrita a seguir) um documento XML onde descreve o componente de forma genérica: a extensão deste arquivo será o .ccd (*corba component descriptor*) e ficará associado ao conjunto de arquivos referentes à implementação ou às implementações do mesmo componente.

Descritor tipo csd

O conjunto das implementações e o arquivo descritor do componente podem ser reunidos em um pacote, ao qual se associa outro arquivo descritor, para descrever as diferentes

implementações existentes dentro do pacote de software. Esse arquivo descritor tem como extensão o .csd (*corba software descriptor*) e descreve cada uma das implementações do componente que contém. Em cada descrição da implementação, especifica o nome do descritor do componente (.ccd) associado à implementação. O arquivo descritor de software (.csd) indica o nome do arquivo que contém o código do fabricante do componente (*component home executor*): opcionalmente especificará o endereço do arquivo, se o lugar é outro diferente ao local; além de indicar o nome do comando para executar o programa do arquivo anteriormente mencionado, o qual é definido como ponto de entrada (*entry point*) e utilizado no processo de montagem do componente. Através deste ponto de entrada será possível ativar, durante o processo de montagem, uma instância do fabricante do componente, para que a partir daí possam ser criados os componentes. Também podem-se incorporar, opcionalmente, um conjunto de arquivos de propriedades.

Descritor tipo cad

Para preparar um pacote com um grupo de componentes, acrescenta-se um arquivo descritor da forma de agrupar os componentes, com a extensão .cad (*component assembly descriptor*) que contém, entre outros, a identidade dos componentes e as ligações entre eles.

Arquivo tipo aar

Finalmente, os autores sugerem empacotar em um arquivo, com formato ZIP, os pacotes anteriores (pacote com uma implementação ou um pacote com um conjunto de implementações) com a extensão .aar

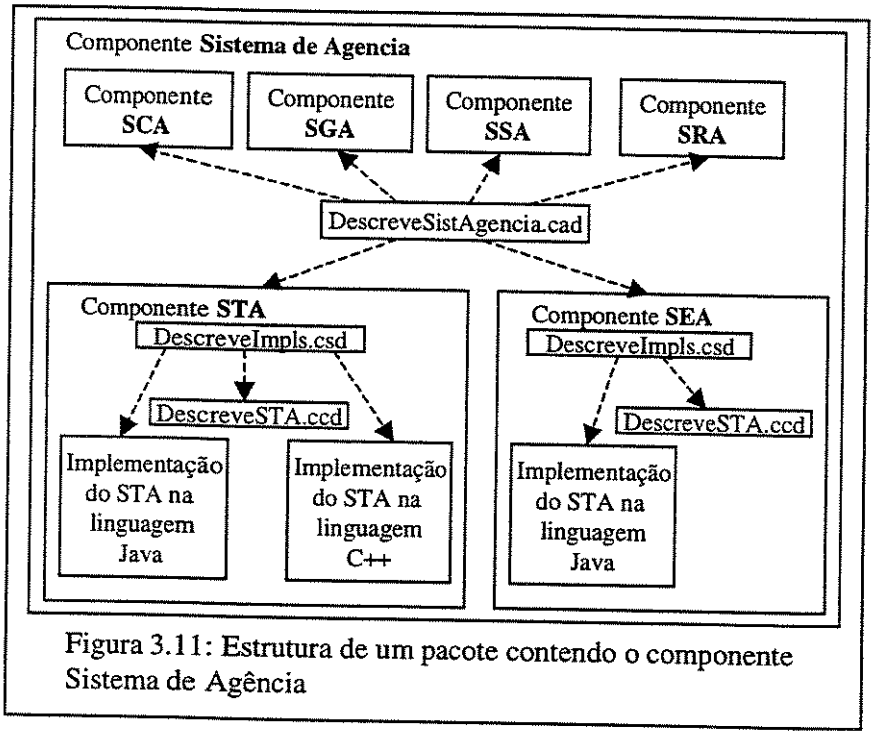

Figura 3.11: Estrutura de um pacote contendo o componente Sistema de Agência

Exemplo

Na figura 3.11, apresenta-se um exemplo de uma aplicação, nomeada Sistema de Agência, constituída pelos componentes: Serviço de Execução de Agentes (SEA), Serviço de Transporte de Agentes (STA), Serviço de Comunicação entre Agentes (SCA), Serviço de Gerenciamento de Agentes (SGA), Serviço de Registro de Agentes (SRA) e Serviço de Segurança de Agentes (SSA). Os nomes destes componentes e a relação entre eles estão descritos no arquivo "DescreveSistAgencia.cad". Cada componente agrupa seus arquivos

correspondentes ao código das diversas implementações feitas. Por exemplo, o componente STA é constituído por:

- O arquivo "DescreveSTA.ccd" que descreve o STA independente da implementação.
- O arquivo "DescreveImpls.csd", indicando que se trata de um componente com duas implementações, uma em Java e a outra em C++, declarando o arquivo "DescreveSTA.ccd" como o descritor geral deste componente.

Logo, o Sistema de Agência pode distribuir-se como um componente, deixando que a ferramenta faça o desempacotamento, instalação e execução de cada um dos componentes.

3.1.7.7.CIDL (*Component* IDL)

O documento propõe utilizar uma nova linguagem para descrever um componente e sua implementação, para que a partir dele possam se obter, através de um compilador CIDL, os esqueletos de programas na linguagem alvo, necessários para implementar o componente descrito. O implementador é assim apoiado na geração de todos os arquivos necessários para a implementação do componente, a um nível de abstração superior ao de IDL. O compilador CIDL deverá ser capaz de gerar, pelo menos parcialmente, os arquivos descritores de componentes.

Para isso, define-se o conceito de composição (*composition*), como uma unidade de implementação do componente, o qual deve conter a seguinte informação:

- Tipo do fabricante (*home*) de componentes.
- Tipo do fabricante do esquema de armazenamento que o componente utilizará.
- Especificação do nome do executor do fabricante (*home executor*), o qual é utilizado para nomear o arquivo-*skeleton,* gerado pelo compilador CIDL para implementar o fabricante de componentes.
- Especificação do nome do executor ou *component executor*, o qual é utilizado para nomear o arquivo-skeleton gerado pelo compilador CIDL para implementar o componente. Dentro do executor, opcionalmente, pode-se especificar os segmentos, ou seja, especificar a forma de agrupar os diferentes objetos (*facets*) que compõem o componente.

Como exemplo apresenta-se a seguinte especificação CIDL:

```
// Arquivo ComercioEletrônico.cidl
import ::AgentesMóveis;
module ComercioEletrônico{

        composition session ImplCompras {
                home executor ImplFabricaAgenteCompra {
                        implements AgentesMóveis::AgenteCompra;
                        manages ImplSessãoCompras;
                };
```

```
    };
};
```

a qual está indicando que se trata de uma composição para um componente do tipo *Session*, cujo nome é "ImplCompras". Tem um *home executor* que será nomeado como "ImplFabricaAgenteCompra" (nome do artefato de programação que vai conter a implementação do fabricante do componente). A composição implementa o componente nomeado agora como "AgenteCompra" e definido em IDL no módulo importado, nomeado "AgentesMóveis". O artefato de programação para o componente (ou seja, o programa que vai implementar o componente na linguagem alvo) será nomeado "ImplSessãoCompras".

O módulo "AgentesMóveis" contém as seguintes interfaces IDL:

```
// Arquivo AgentesMóveis.idl
module AgentesMóveis {
        interface Gerenciamento {
                boolean itinerario (in string endereço)
        };

        interface ProgramaCompra {
                void comprar(in string nome);
        };

        component AgenteCompra{
                provides Gerenciamento gerencia;
                provides ProgramaCompra comprador;
        };
};
```

O módulo "AgentesMóveis" define um componente que implementa duas interfaces (gerencia e comprador) cujos tipos são especificados pelas interfaces "Gerenciamento" e "ProgramaCompra" respectivamente.
Na especificação IDL do módulo "AgentesMóveis", aparece o termo *component e provides*, os quais são uma extensão ao vocabulário atual do IDL.

No arquivo "ComercioEletrônico.cidl" especifica-se, de forma genérica (independente da linguagem de programação), como se vai implementar o componente. Neste caso, o compilador CIDL pegará os arquivos "ComercioEletrônico.cidl" e "AgentesMóveis.idl" e gerará automaticamente na linguagem alvo de programação:

- Todas as operações que um componente do tipo *Session* deve ter.
- As interfaces próprias do componente especificado no módulo "AgentesMóveis"

Considerando Java como a linguagem alvo, serão gerados os arquivos (*artifacts*) *skeletons* para o executor do componente, nomeado "ImplSessãoCompras.java" e para a implementação completa do *home executor*, nomeado "ImplFabricaAgenteCompra.java". O implementador do componente deverá preencher, com código adicional (das operações), o

arquivo "ImplSessãoCompras.java". Adicionalmente gerará os arquivos com código correspondente à implementação das interfaces especificadas no arquivo "AgentesMóveis.idl", como normalmente é feito pelos compiladores IDL tradicionais.

O compilador CIDL deverá ser capaz de gerar parcialmente o arquivo descritor de componentes (.ccd), onde o implementador do componente procederá ao complemento da informação, especialmente a referente à descrição do tipo de *container* que deverá ser instanciado no lugar, para executar o componente.

Durante a montagem do componente, será instanciado um *container* baseado na informação contida nos arquivos descritores, especialmente no ccd, o qual é o adequado para o componente que está sendo instalado.

3.1.7.9.Comentários

Em resumo, através desta especificação, é possível construir ferramentas de software que permitem desenvolver componentes e manipular seu ciclo de vida. Além disso, viabiliza-se o desenvolvimento de aplicações, montando componentes implementados por diferentes fabricantes, onde é possível controlar as diversas etapas, desde sua criação, instalação, execução e eliminação.

Esta especificação é importante para a mobilidade dos agentes porque aborda a padronização de uma entidade mais abrangente, o componente, e a padronização de um ambiente de execução. Estas duas questões são aplicáveis aos requisitos dos agentes móveis, que precisam de uma estrutura e ambiente de execução padronizados, para que possam se movimentar por lugares normalmente heterogêneos e para que os usuários possam utilizar sistemas de agentes de diferentes fabricantes.

3.2. Mobilidade em Java

Desde maio de 1995, quando foi apresentada pela Sun Microsystems, a linguagem Java [Java96] tem tido um crescimento vertiginoso, convertendo-se em uma plataforma adequada para desenvolver aplicações em rede. É uma linguagem de programação orientada a objetos que possui, inclusive, características próprias de mobilidade, fornecidas pelos *applets*, carregados pelos *browsers*, através das classes que o compõem, a partir de qualquer servidor espalhado na rede. Também é altamente portável porque está baseada em uma máquina virtual, permitindo que seja compilado em um código intermediário, nomeado *bytecode*, a ser transportado para qualquer lugar que tenha implementado a máquina virtual para Java. Aí será finalmente convertido para o código da máquina alvo e executado.

Assim, a mobilidade e a portabilidade são as características mais importantes desta linguagem, além da facilidade de seu aprendizado.

A maioria dos projetos de agentes móveis estão sendo desenvolvidos em Java, devido às características anteriormente mencionadas. Este é um requisito fundamental na programação dos agentes móveis, os quais têm que se movimentar através de ambientes heterogêneos, tanto em relação ao hardware quanto em relação ao sistema operacional. Além disso, Java fornece facilidades para incorporar segurança nas aplicações, a possibilidade de serializar um objeto, e um meio de comunicação distribuída com RMI.

Sem dúvida que esta é hoje a linguagem mais importante, com estas características. Com certeza outras linguagens irão aparecendo no cenário, como é o caso de Oberon [Pfister98], que é uma linguagem Pascal modificada para ser pre-compilada para "byte code" e assim poder-se executar por uma máquina virtual. De qualquer forma, Java está se convertendo em algo mais que uma linguagem: a partir dela estão-se, por um lado, construindo plataformas que oferecem facilidades para sistemas distribuídos orientados a objetos, por outro desenvolvendo arquiteturas para suportar os conceitos de componentes de software. Hoje, as aplicações da Internet estão sendo desenvolvidas principalmente em Java, e como a Internet/intranet é a vitrina onde as atividades do ser humano cada vez mais estão se incorporando (comércio eletrônico, workflow, busca de informação, trabalho cooperativo, telecomunicações, comunicação multimídia, etc.), não se poderá subestimar a importância de Java, também no desenvolvimento dos agentes móveis e de suas aplicações nas áreas mencionadas.

A seguir, serão apresentados e discutidos alguns elementos mais relevantes da plataforma Java, sem descrever como eles funcionam, para analisar as facilidades que oferece na implementação de um ambiente para suportar agentes móveis.

3.2.2. JavaBeans

Tendo como principal objetivo o desenvolvimento rápido de aplicações baseadas nesta linguagem, foi especificada a estrutura de um componente que foi nomeado *JavaBeans* [JavaBeans99]. Esta especificação permite implementar componentes (*JavaBeans*) que se podem manipular por ferramentas visuais, da mesma forma que outras ferramentas como *Visual Basic*. Estes componentes interagem através de eventos e suas propriedades podem ser definidas pelo usuário, permitindo construir aplicações baseadas na filosofia dos componentes.

3.2.3. Enterprise JavaBeans

Apesar de Java apresentar um sistema altamente portável e de se fornecerem ferramentas para implementar aplicações baseadas em componentes visuais, a portabilidade e a interoperabilidade entre as aplicações desenvolvidas por diferentes fabricantes não estava resolvida. Para tal, foi especificado um conjunto de APIs que padroniza especialmente um componente para ser manipulado por ambientes de execução de diferentes fabricantes, padronizando também o acesso aos ambientes de execução.

A estrutura destes componentes, nomeados "*enterprise java beans*" ("ejb") [EJB99] está baseada nos *JavaBeans*. Os "ejb" podem-se empacotar sob uma estrutura que é descrita, utilizando arquivos descritores, em XML. Assim, pode-se construir uma aplicação constituída por vários componentes e onde a relação entre eles também está descrita nos arquivos descritores, de tal forma que ferramentas automatizadas possam manipular o pacote e proceder à instalação dos componentes, ativar suas ligações e executar a aplicação. Desta forma, conseguiu-se fornecer os elementos que permitem desenvolver ferramentas automatizadas para construir aplicações baseadas em componentes, onde não somente poderão se manipular componentes visuais, como os *JavaBeans*, como também objetos servidores que poderão trabalhar em segundo plano (*background*).

A especificação dos "ejb" foi a base para a especificação dos Componentes CORBA: de fato os "ejb" são considerados como exemplo de um Componente CORBA.

Por outro lado, os "ejb" podem tirar proveito de todos os recursos que a plataforma Java fornece, por exemplo, fazer uso das APIs de segurança, utilizar RMI (*Remote Method Invocation*) [RMI99] junto com JNDI (*Java Naming and Directory Interface*) [JNDI99] para desenvolver aplicações distribuídas e usar JDBC (*Java Database Connectivity*) [JDBC99] e JTA (*Java Transaction API*) [JTA99] para acessar bancos de dados com controle das transações. Podem acessar servidores escritos em outras linguagens ou ser acessados por clientes em outras linguagens, desde que utilizem a interface Java-IDL e IDL-Java respectivamente.

Em resumo, a plataforma Java oferece os mecanismos necessários para implementar uma plataforma que suporte agente móveis, salientando-se as seguintes características:

- A linguagem oferece uma alta portabilidade comparada com outras linguagens, sendo adequada para a implementação dos agentes que se movimentam pela rede.
- Com o uso dos "ejb", a portabilidade e interoperabilidade ficam resolvidas nesta plataforma. Assim, a execução dos agentes fica garantida em qualquer ambiente Java que visitem.
- Permite o carregamento dinâmico das classes, unidade mínima para carregar a partir de qualquer lugar de uma rede. Esta é uma característica própria da linguagem, a qual pode-se explorar para movimentar os agentes. Contudo, como os agentes não necessariamente estarão escritos em Java, esta característica não pode ser estendida para as outras linguagens, sendo necessário implementar outros mecanismos para transportar o código.
- É possível construir aplicações distribuídas.

- Os *applets* permitem movimentar o código unicamente a partir do servidor para o *browser*. Pode-se dizer, que Java oferece um elemento de *software* (o *applet*) com mobilidade restringida.
- A interface com CORBA está especificada, permitindo que aplicações em Java possam comunicar-se com aplicações escritas em outras linguagens.
- Oferece APIs especializadas para incorporar a segurança na transmissão das classes e incorporar funcionalidades de segurança nos programas, como criptografia e verificação.

3.3. Discussão dos aspectos da mobilidade considerando CORBA e Java

Pelas características mencionadas, a linguagem Java está sendo utilizada nos diversos projetos, permitindo desenvolver rapidamente plataformas para o suporte de agentes móveis. Características como portabilidade, interoperabildade e mobilidade do código são muito importantes na implementação dos agentes móveis, pelo menos assegurando que agentes escritos em Java possam ser executados em diferentes ambientes de *hardware* e sistemas operacionais. Por outro lado, é factível que agentes em Java se possam comunicar com entidades escritas em outras linguagens.

Portanto, a plataforma Java tem todos os elementos necessários para implementar agentes móveis. Contudo é necessário padronizar a estrutura dos mesmos, como do ambiente de execução, para que a mobilidade não fique restringida aos agentes e ambientes em Java. Para ter uma plataforma universal de suporte de agentes, baseada só em Java, é necessário padronizar a forma usada pelos agentes para viajar através das diferentes agências. Em tal caso, acha-se que CORBA é mais adequado para especificar uma plataforma para o suporte universal de agentes móveis, devido à sua característica integradora de *software*, escrito em diferentes linguagens e por ser uma arquitetura que permite padronizar o protocolo de interação entre as diversas entidades participantes, independentemente do tipo de plataforma e linguagem.

Java é a linguagem mais adequada, até agora, para implementar os aspectos de portabilidade e interoperabilidade de um agente. Outros aspectos da plataforma podem ser implementados mais eficientemente em outras linguagens, especialmente aqueles que ficam estáticos na agência. Desta forma, a implementação não fica restrita a uma linguagem específica e não dependerá das versões das diferentes linguagens que as utilizam.

Para efeitos de mobilidade de objetos CORBA, é importante salientar que ainda existem problemas a serem resolvidos, em particular em relação à portabilidade das implementações, através do POA (a nível de código fonte). Por exemplo, permanece a questão da não portabilidade das referências de objetos CORBA. Se um cliente "A" tem a referência para um objeto "X", se este objeto é movimentado para outro local, onde o ORB pertence a outro fabricante, esse ORB não poderá resolver a referência da requisição do

cliente "A" porque não sabe como foi empacotado pelo outro fabricante. Este caso haverá que procurar uma solução a nível da aplicação. Por exemplo, antes de mover-se, informa-se o cliente para fazer uma desligação ordenada: ao chegar ao destino o cliente deve ser informado do novo endereço ou entregando-lhe a nova referência. Esta forma de solução quebra o modelo do cliente/servidor. Melhor seria fazê-lo a nível dos ORBs envolvidos na transação, o que significaria modificar a especificação do CORBA. A solução definitiva parece passar pela padronização definitiva da referência dos objetos CORBA.

CORBA e Java podem-se considerar como duas ferramentas que se complementam para construir uma plataforma universal que suporte agentes móveis. A primeira para especificar uma arquitetura padrão e a segunda para implementá-la ou para implementar aqueles aspectos relacionados com a portabilidade e interoperabilidade dos agentes: garante-se a possibilidade de implementar parte da plataforma utilizando outras linguagens que permitam tirar um maior proveito do ambiente de execução. Além da possibilidade de construir ferramentas automatizadas para gerenciar o ciclo de vida de um agente; neste sentido, a OMG está avançando na especificação de um serviço nomeado *Meta Object Facility* [MOF97], que permite especificar a forma de descrever objetos (meta-dados).

A especificação do POA pela OMG permite implementar objetos CORBA que sejam portáteis (a nível do código fonte) através de diferentes fabricantes de ORBs. Este fato melhora a portabilidade dos agentes, se eles são especificados em CORBA

Muitos problemas permanecem, contudo, sem solução.. Por exemplo, a questão de salvar o estado de um agente de uma forma global, ainda não foi esclarecida satisfatoriamente. No caso de Java, o estado é tratado a nível do objeto (Serialização) [JavaSerial98]. No caso de CORBA é tratado a nível de objeto CORBA (Externalização) [Externaliz97] e Persistência [PSS99]). Mas ainda não foi esclarecido como tratar o estado de um agente, formado por um objeto, por um conjunto de objetos, por um componente ou por programas não orientados a objetos. Em CORBA, está sendo especificado um serviço para passar objetos por valor, que ainda apresenta problemas de implementação.

Tanto a especificação do Componente CORBA como EJB podem ser a base para definir uma estrutura padrão de um agente móvel e a partir dai, abordar a questão do estado de um agente.

Qualquer implementação a partir de uma especificação CORBA é mais abrangente que uma baseada em uma linguagem, ainda que essa linguagem seja Java. Primeiro, porque é uma especificação, permitindo uma implementação independente da linguagem (desde que se tenha definido o mapeamento IDL) e segundo, porque é uma especificação estabelecida por um grupo importante de empresas no mundo da informática. Logo, o desenvolvimento de uma arquitetura para o suporte de agentes móveis, baseado em CORBA, apresenta grandes vantagens para os usuários, que teriam mais opções na escolha do produto mais adequado às suas necessidades, e interoperável com outros produtos similares.

3.4. Resumo

Neste capítulo analisou-se como OMA/OMG e Java podem suportar a mobilidade dos agentes. No primeiro caso, analisaram-se os serviços CORBA *LifeCycle*, Externalização, Nomes e descreveram-se os principais aspectos da proposta MASIF e do modelo Componentes da OMG. Foram também descritos alguns aspectos de GIOP e POA da especificação do CORBA.

Concluiu-se que o serviço de *LifeCycle* não é adequado para suporte à mobilidade de agentes, porque exige que o programador da aplicação se preocupe em implementar a operação "move" do serviço e porque o código dessa operação deve ser parte do agente. Se o agente é composto de vários objetos Corba, então a movimentação de cada objeto deve ser seqüencial, o que pode degradar o desempenho do agente.

Em relação ao serviço de Externalização, achou-se adequado utilizá-lo para salvar o estado do agente. Contudo esse serviço não padroniza a forma de ordenar os dados no *stream*, sugerindo usar o CDR como formato padrão.

O serviço de Nomes é adequado para registrar as agências, no caso de estas serem objetos Corba. Mas, para registrar os agentes, é melhor definir um outro serviço, associado à agência, que seja capaz de registrar agentes que podem ou não ser objetos Corba. Contudo não se descarta seu uso no escopo local da agência. O fato de os agentes mudarem freqüentemente de lugar e de possuirem tempos de vida variáveis, torna inadequado o uso de serviços de nomes globais para registrar os agentes, pela possibilidade de se gerar um tráfego indesejável na rede.

Descreveu-se o MASIF, destacando como sua principal função a padronização das agências, para receber e despachar agentes de qualquer fabricante.

Em seguida descreveu-se como GIOP permite aos fabricantes de ORBs gerenciar, de forma transparente à aplicação, a referência dos objetos que se movimentam.

Explicaram-se também as principais funcionalidades do POA e como ele pode ajudar na portabilidade dos agentes, quando eles se movimentam através de agências implementadas sobre ORBs de fabricantes diferentes. Esta portabilidade pode ser melhorada, considerando o modelo de Componentes, o qual permitiria padronizar aspectos de criação, instalação, execução e eliminação dos agentes (*deployment*).

Finalmente, discutem-se brevemente os JavaBeans e Enterprise Java Beans, no sentido como estes mecanismos podem ajudar a implementar plataformas para o suporte de agentes móveis. De fato, a OMG baseou-se nestes modelos para especificar o modelo de Componentes.

4. Aspectos de transporte em projetos de agentes móveis

Atualmente existem inúmeros projetos acadêmicos como também alguns produtos comerciais que fornecem plataformas para o suporte de agentes móveis. Entre os projetos de pesquisa encontram-se, por exemplo: D'Agents, Ara [Peine97], Mole [Strasser96], Moa, Telescript [White94], Java-to-go [Li96], Gypsy [Lugmayr99], etc. Entre os produtos comerciais temos entre outros: Aglet, Grasshopper, Jumping Beans, e Voyager.

Uma vez que o objetivo deste trabalho é apresentar um serviço de transporte de agentes, nas próximas seções serão descritos vários projetos, considerando-se as formas de abordagem das questões de nomeação e de endereçamento dos diversos tipos de objetos participantes (agências e agentes), os protocolos de transporte oferecidos e, formas de empacotamento do agente e de outros dados e meta-dados, antes de ele ser despachado para o destino. São estes os aspectos mais relevantes que podem levar à comparação com a proposta deste trabalho de tese.

No final do capítulo será apresentado o projeto CoABS, ainda em andamento, baseado na tecnologia Jini, explicando-se o seu escopo e objetivos, salientando a importância que possa ter na aplicação de agentes móveis.

4.1. D'Agents

Este é um projeto de pesquisa que está sendo desenvolvido na universidade da Darmouth College, [Gray95]. Os autores definem uma arquitetura para o suporte de agentes móveis escritos originalmente em Agent Tcl [Gray97], que é uma extensão da linguagem Tcl ("Tool command language") para suportar migração (agent_jump), clonagem (agent_clone) e réplica (agent_submit) de um agente. A migração é realizada mantendo o programa "script" em um "stack" de execução, quando migrar, salvando-se a seqüência de comandos do "stack" e o indicador do comando seguinte a ser executado, e a ser recuperado no destino [Kotz97]. Inicialmente o nome do projeto era AgentTcl, mas como agora está se incorporando o suporte de outras linguagens como Java, Scheme e Python o nome foi alterado para D'Agents.

O nome de um agente é composto pelo nome da máquina que aceita o agente (incluindo o domínio), o seu número IP, o número do agente que o identifica univocamente nessa máquina e um nome simbólico local. Portanto, toda vez que o agente muda de local, seu nome também mudará.

Para manter a informação sobre o endereço original do agente (onde ele foi criado), o agente leva consigo o nome que tinha na origem, o qual é conhecido como "root". Para isso, define-se um "agent array", atualizado toda vez que o agente muda de local.

Os programas do agente são instalados em um diretório único de execução, mas a documentação do projeto, disponível publicamente, não dá mais detalhes.

No momento de especificar a sentença de migrar, o nome do programa que contém o "script" de comandos é indicado dentro da sentença, incorporando-se também as variáveis e procedimentos dentro desse programa que o agente pode acessar. Por exemplo:

agent_submit $machine –vars number –procs factorial –script {factorial $number}

Neste caso um agente vai ser criado na máquina indicada pela variável "machine", onde o programa do agente é o "script" nomeado "factorial" que atua sobre a variável "number". Para isso a sentença está autorizando o uso da variável "number" e o procedimento "factorial" através das opções "vars" e "procs".

Os agentes comunicam entre si, na Internet, através de TCP/IP.

A arquitetura é implementada por dois componentes:

1. Um interpretador para as linguagens Tcl, Java, Scheme ou Python;
2. Um servidor que executa os agentes que recebe, manipula as mensagens entre os agentes e monitora os agentes que estão sendo executados na máquina.

4.2. Aglet

É um sistema de agentes [Aglet97] baseado em Java, desenvolvido pela IBM. Este sistema permite criar, migrar e clonar agentes, alem da sua eliminação. Também permite que os agentes possam se comunicar via mensagens. Para isso, define-se um serviço de agentes (agência) encarregado de receber e despachar os agentes, além de outras funções e facilidades, como segurança e troca de mensagens entre agentes.

4.2.1. Identificação

O serviço de agentes e os agentes usam o mesmo esquema URL para sua identificação. Por exemplo, para se identificar um serviço específico de agentes pode ser usado o seguinte URL:

atp://joe.trl.ibm.com
atp://joe.trl.ibm.com:10434

Para identificar um agente, usa-se o endereço do serviço de agentes, onde o agente está em execução, mais um nome simbólico que representa o grupo de agentes ao qual pertence, e um string de caracteres alfanuméricos que identificam univocamente o agente dentro do serviço de agentes, separado pelo símbolo "#".
Exemplo:

atp://dca.fee.unicamp.br/alunos#ag01234

No caso trata-se de um agente com identificação "ag01234", o qual se encontra em execução dentro do grupo de agentes nomeado "alunos", na máquina com o endereço"dca.fee.unicamp.br". O valor "ag01234" é único dentro do serviço de agentes referenciado. Logo se o agente migrar, sua identificação mudará, porque o serviço de agentes que o receber, definirá outro identificador para ele.

Para o transporte dos agentes define-se um protocolo a nível da aplicação, nomeado ATP (Agent Transfer Protocol). Este protocolo pode transportar agentes escritos em qualquer linguagem, definindo vários formatos baseados em mensagens do tipo "request" e "response"[Atp97]. Basicamente padronizam-se um cabeçalho e um corpo da mensagem que contém o código do agente e do seu estado (serializado no caso de um agente em Java): este código poderá ser comprimido.

4.2.2. Formato das mensagens do tipo Request

"dispatch"| "retract"| "fetch"| "message"	Recurso_URI	Versão_ATP
"Date:" Data e hora		
"User-Agent:" <Descrição textual>		
"From:" Endereço eletrónico		
"Agent-System:" <Descrição textual>		
"Agent-Language:" <Descrição textual>		
"Content-Type:" <Descrição textual>		
"Content-Encoding:" <Descrição textual>		
"Content-Length:" <Número decimal>		
"Agent-Id:" <Serie alfanumérica>		
Corpo da mensagem		

Exemplo:

dispatch	/agentes_comerciais	ATP/0.1
Date: "Sat, 22 Apr 1999 10:15:00 GMT"		
User-Agent: Tahiti Aglet viewer		
From: Dono@dca.fee.unicamp.br		
Agent-System: ibm.aglets		
Agent-Language: java		
Content-Type: aglet		
Content-Encoding: zip		
Content-Length: 7845		
Agent-Id: ag01234		
Corpo da mensagem		

No exemplo a mensagem "request" é do tipo "dispatch" para despachar um agente incorporado ao conjunto de agentes nomeado de "agentes_comerciais". O protocolo utilizado é o ATP versão 0.1 e o agente está sendo despachado com data: Sábado 22 de Abril de 1999, às 10:15:00 Hrs. GMT. O agente foi gerado pela aplicação "Tahiti Aglet viewer" e o endereço do mail do dono do agente é "dono@dca.fee.unicamp.br". O agente pertence às plataformas do tipo "ibm.aglets", está escrito em Java e é do tipo "aglet". O

corpo da mensagem está comprimido com formato zip e tem um comprimento de 7845 bytes; sua identificação é "ag01234".

No caso de uma mensagem "request" do tipo "retract", os campos "Agent-Language:", "Content-Type:", "Content-Encoding:", "Content-Length:" e o corpo da mensagem, não precisam ser especificados porque está se solicitando recolher um agente identificado pelo "Agent-Id". Neste caso, o campo "Recurso_URI" especifica o nome do agente a ser recolhido ("retracted").

No caso de uma mensagem "request" do tipo "fetch" os campos "Content-Type:", "Content-Encoding:", "Content-Length:", "Agent-Id" e o corpo da mensagem, não precisam ser especificados por que está-se solicitando carregar o código de uma classe. Neste caso, o campo "Recurso_URI" especifica o nome do arquivo e seu diretório que contém a classe.

Finalmente, no caso de uma mensagem "request" do tipo "message", todos os campos são especificados: o campo "Recurso_URI" identifica o agente receptor da mensagem.

4.2.3. Formato das mensagens do tipo Response

Versão_ATP	Código da resposta	Explicações
"Date:" Data e hora		
"User-Agent:" <Descrição textual>		
"Agent-System:" <Descrição textual>		
"Agent-Language:" <Descrição textual>		
"Content-Type:" <Descrição textual>		
"Content-Encoding:" <Descrição textual>		
"Content-Length:" <Número decimal>		
"Agent-Id:" <Serie alfanumérica>		
Corpo da mensagem		

O campo "Código da resposta" é um numero de 3 dígitos que expressa o resultado de um "request". E o campo "Explicações" é um texto curto com a descrição do código da resposta. O resto é similar às mensagens "request".

4.3. Grasshopper

Este é um produto comercial que está sendo desenvolvido pela empresa IKV++ e o Instituto FOKUS de Berlim [Grasshopper99]. Consiste na implementação de um ambiente distribuído para a execução de agentes móveis. A implementação é totalmente feita em Java e adere à especificação do MASIF.

A plataforma está baseada nas agências que suportam as facilidades para a mobilidade dos agentes (incluindo seu ciclo de vida completo), os quais são "threads" executados em grupos de "threads" (implementação do conceito "place" do MASIF).

Cada agente tem incorporado os métodos "move()" e "copy()", os quais, quando são invocados pelo mesmo agente, farão a ação de migrar ou copiar o agente, respectivamente, para a agência destino especificada como parâmetro de endereço em um formato URL.

A transferência do agente é feita através dos serviços de transporte e de comunicação fornecidos pela agência . Estes serviços operam em forma hierárquica, ou seja, através da operação "move()" ou "copy()", o serviço de transporte é contatado, que procede à externalização do agente (no caso de migração) e à coordenação da transferência efetiva do agente, é feita pelo serviço de comunicação.

O serviço de comunicação está encarregado de todas as interações remotas que acontecem entre as diversas partes que compõem o ambiente distribuído de agentes do Grasshopper, tais como comunicação entre agentes de forma transparente com relação a localização, transporte dos agentes e localização de agentes via registro de regiões. Para a transferência, o serviço suporta os seguintes protocolos:

- Socket com e sem SSL
- RMI com e sem SSL
- CORBA (Visibroker)

No caso de usar "Sockets" ou RMI, o agente tem que ser transferido. Isto é feito através de um "ClassLoader", que transferirá os arquivos necessários com o código das classes para a execução do agente.
No caso de CORBA, usam-se as operações especificadas pelo MASIF: "fetch_class()" e "receive_agent()".

Todos os componentes que sejam criados (agências, agentes e places) devem ser registrados no Registro de Região. Quando um agente muda de região, durante sua viagem, ele deve registrar-se quando chegar à nova região, indicando com qual agência está associado nessa região.

O administrador da agência pode gerenciar todos os agentes que estão-se executando na mesma. Ele pode criar, salvar, eliminar, suspender ou reiniciar um agente, e é o dono do agente durante toda a vida do agente. O agente terá um único identificador (dentro de uma região) e poderá ter um nome associado.

4.4. MOA (Mobile Objects and Agents)

Este é um projeto [Milojicic98a] desenvolvido no Open Group Research Institute para suportar a migração, comunicação e controle de agentes. Além do projeto anterior, os autores estudavam o uso de JavaBeans [Milojicic98] como referência de um modelo de componentes, para a configuração e construção de sistemas de agentes e aplicações baseadas em agentes móveis. O projeto também adere à especificação MASIF para que o sistema desenvolvido possa interoperar com outros sistemas de agentes.

O projeto está sendo desenvolvido totalmente em Java aproveitando as facilidades que oferece para implementar: as propriedades, heterogeneidade, segurança (limitada), serialização de objetos, carregamento dinâmico das classes e multithreading.

A arquitetura do ambiente de agentes (agência) tem um módulo, nomeado Negotiator, encarregado de negociar o agente antes de sua visita; sobre ele, está o modulo Mover encarregado de gerenciar a negociação da migração, externalizar o agente e transferi-lo para o destino.

Antes de migrar, os arquivos do agente, mais outros arquivos contendo as propriedades, interfaces para acessar a agência e as interfaces para monitorar o agente, são salvos em um arquivo do tipo JAR (JAva aRchive), o qual é transferido para o destino usando o mecanismo do "ClassLoader" do Java.

O sistema implementa seu próprio Servidor de Nomes para registrar os diversos componentes que constituem o sistema: agentes (AgentName), ambiente de agentes (AEname), "places" (PlaceName) e servidores de nomes (ServerName). O formato destes nomes é o seguinte:

AEname (ae):	h:p	Onde:
ServerName:	h:p	h: nome do host
AgentName (a):	ae_home#f_l.g	p: nome da porta
PlaceName:	a_owner%ae_residing	f: nome da família
		l: nome do envio

Como o formato indica, a sintaxe para os nomes do ambiente de agente e do servidor de nomes é a mesma, especificando o nome do hospedeiroonde está residindo e o número da porta por onde "escuta" os requerimentos.

O nome do agente é composto pelo nome do ambiente onde o agente foi criado, o símbolo separador "#", o nome da família (f) do agente, junto com um número (l) que o identifica dentro da família. No caso, de ser uma cópia de um outro agente (clonagem) acrescenta-se um número indicando o nível que tem dentro da família, separado pelo símbolo ".".

O nome do "place" é composto pelo nome do agente que contém, mais o nome do ambiente do agente onde está residindo, separados pelo símbolo "%". Portanto, neste projeto, um "place" só pode conter um agente, servindo de representante ("proxy") do agente quando deixa o lugar, além de cumprir funções de segurança e de ser o canal para fornecer os recursos para o agente.

A mobilidade é baseada em mensagens, onde o conteúdo da mensagem é um arquivo tipo JAR, que nomeado de "bucket", o qual contém o agente e seus recursos (serialização, propriedades, "classloader", informação sobre seus canais de comunicação e outros). Quando a mensagem chega no destino, o ambiente de agentes no local carrega o agente usando o "classloader" incluído na mensagem associada ao agente.

Para a transferência de um agente, via mensagens, o ambiente de agentes utiliza os "sockets", através do qual a origem solicita primeiro um reconhecimento do destino para enviar o agente: caso positivo procede-se ao envio do pacote que contém o agente.

4.5. JumpingBeans

É um produto comercial desenvolvido pela empresa Ad Astra Enginnering Inc. [AdAstra99] com as seguintes características:

- É implementado totalmente em Java;
- O modelo é baseado na arquitetura cliente/servidor, onde os clientes são as agências. O servidor, além de ser responsável pela segurança do sistema, é a entidade encarregada de administrar e monitorar as diversas aplicações móveis que estejam ativas nas agências, ligadas com o servidor;
- Este produto permite a mobilidade de qualquer aplicação Java (não necessariamente agentes ou JavaBeans) que seja serializável e que implemente a interface *com.JumpingBeans.core.api.MobileApp* fornecida;
- Neste modelo, as aplicações móveis quando viajam para outra agência, devem passar primeiro pelo servidor onde serão inspecionadas, por questões de segurança: posteriormente o servidor envia a aplicação para a agência destino;
- Se uma aplicação móvel ou um processo quer se comunicar com outra aplicação móvel, enviando-lhe uma mensagem, primeiro a mensagem passa pelo servidor, onde é inspecionada, para posteriormente enviá-la à agência onde se encontra a aplicação móvel;
- A trajetória da aplicação está baseada em um itinerário que as agências utilizam para despachar a aplicação;
- No caso do transporte, cada agência atua, através de um "daemon", escutando em uma porta TCP/IP e esperando pela recepção de uma aplicação móvel. O uso de RMI para seu traslado não é especificado, mas tudo que está relacionado com a aplicação (código, estado e dados) é empacotado e transmitido para o destino. Este pacote não é desserializado no servidor mas sim na agência destino;
- Não adere ao MASIF;
- A segurança é implementada usando os mecanismos fornecidos pela linguagem Java;
- Não exige nem define um formato específico para nomear a aplicação móvel.

Assim o modelo cliente/servidor implementado gera uma plataforma centralizada para aplicações móveis, que pode ser útil para sistemas de pequeno porte mas, claramente, deficiente em redes de grande porte e de alto tráfego, porque a passagem da aplicação móvel de uma agência para outra sempre é feita através do servidor. Como compensação, sua implementação permite ter agências leves, porque elas não devem preocupar-se com questões de segurança nem de administração do sistema.

4.6. CoABS e Jini

CoABS (Control of Agent Based Systems) [CoABS00] é um projeto suportado pelo Departamento de Defesa dos Estados Unidos, no qual é implementado o conceito de "Agent Grid", utilizando-se a tecnologia Jini como plataforma. O objetivo do projeto é desenvolver um ambiente para a instalação, configuração, execução de todo tipo de agentes, assim como a comunicação entre eles. Tenta-se unir o mundo dos agentes inteligentes e estáticos com o dos agentes móveis. É importante salientar-se que este projeto está começando seu desenvolvimento, ainda não sendo muito claras sua arquitetura final e implementação. Achou-se relevante falar sobre este projeto pelo impacto que possa ter sobre o desenvolvimento dos agentes móveis, no futuro.

Na seqüência, discute-se o conceito "Agent Grid" e posteriormente a tecnologia Jini.

4.6.1. O conceito "Agent Grid"

Este termo está baseado no conceito de "Grid", como um sistema ou conjunto de protocolos, destinado à intercomunicação, de forma relativamente simples, dos diversos dispositivos que pertencem a uma comunidade específica. O objetivo é desenvolver uma grade computacional que forneça serviços que sejam dependentes (de outros serviços), consistentes, básicos e baratos. Exemplos de grades ou redes existentes citam-se:

- A rede elétrica, onde os diversos artefatos: rádios, TV, microondas, etc, podem considerar-se como membros desta grade.
- A rede de telefone, onde os telefones, modems, faxes são os membros desta rede.
- A Internet, onde os computadores e browsers participam nesta rede.

Nestes sistemas, os elementos, ou membros da rede, são incorporados a qualquer momento, de uma forma simples e sem muito conhecimento do usuário sobre a sua configuração.

No caso do "Agent Grid", trata-se de um sistema ou de um conjunto de protocolos com o objetivo de suportar qualquer tipo de agentes, por exemplo: agentes inteligentes, agentes móveis, agentes de interface de usuário, agentes reativos e outros. Este sistema caraterizase por ser heterogêneo e interoperável. Os agentes podem ser incorporados ou desligados a qualquer momento, comunicar-se com outros agentes e realizar seus objetivos.

Atualmente, ainda não existe um consenso geral sobre qual seria o modelo de programação mais adequado para uma ambiente deste tipo. Podem se considerar, por exemplo, ambientes dos seguintes tipos:

- DCE (Distributed Computing Environment)
- CORBA
- Plataforma Java

- Outros

Mas no caso do CoABS escolheu-se a tecnologia Jini, descrita a seguir, para implementar este ambiente.

4.6.2. A tecnologia Jini

Jini [Venners99] é uma plataforma computacional que permite ligar e desligar, a qualquer momento, dispositivos inteligentes, ou seja, dispositivos que possuem um microprocessador e memória. Após ligar-se à plataforma, o dispositivo automaticamente pode-se configurar e autodiagnosticar, registrar seu serviço e ficar pronto para ser utilizado pelos usuários da rede ou outros serviços.

Em outras palavras, Jini implementa o conceito de grade para a interligação de dispositivos inteligentes ("smart devices") ou sistemas embutidos ("embedded systems") através de uma rede de comunicação de dados. Para isso, oferece um conjunto de serviços que implementa o protocolo para a operação dos dispositivos na rede.

Do ponto de vista computacional, Jini é um conjunto de APIs baseado na plataforma Java, onde os dispositivos se comunicam via RMI e objetos Java podem mover-se através da rede com a segurança oferecida pela linguagem Java. A partir destas APIs, poderão construir-se novos serviços, que a Sun Microsystem está promovendo para sua implementação por terceiros.

Quando um dispositivo se liga, automaticamente inicia o processo de busca para saber onde se registrar (processo similar no "booting" de um computador); posteriormente registra-se, oferecendo seus serviços, para que possa ser usado pelos usuários e outros dispositivos da rede. Quando o dispositivo quer usar outros serviços utilizará o serviço "lookup". Os protocolos de busca, registro e "lookup" são parte da plataforma Jini, oferecidos através das APIs.

A tecnologia Jini fornece também mecanismos de suporte às transações e de aluguel de serviços, construindo um ambiente de rede dinâmico. Assim, os dispositivos poderão ser usados e pagos pelo tempo de seu uso.

A filosofia por detrás de Jini é de implementação do conceito "o computador é a rede", permitindo evitar a existência de elementos redundantes, usando-se aqueles que estão disponíveis. Por exemplo, se for necessário ter uma saída visual, poderá usar-se um televisor (inteligente), inserido previamente na rede.

Para tornar realidade esta filosofia, a Sun Microsystem está incorporando a infra-estrutura Jini no microprocessador *picoJava*. Desta forma, os dispositivos inteligentes baseados neste microprocessador, poderão incorporar-se dinamicamente e operar em uma rede de dados

que entenda o conjunto de protocolos Jini. Dentro dos dispositivos que conteriam tecnologia Jini, pode-se citar:

- Impressoras
- Dispositivos armazenadores da informação (como discos)
- PDAs ("Personal Digital Assistants")
- Câmaras digitais
- Telefone celular
- Video cassettes digitais (DVCR)
- Televisores
- Controles industriais
- Outros

Para que seja aceite pelo mercado, a Sun Microsystem criou a "Sun Community Source Licence" (SCSL), a qual abre o código fonte da infra-estrutura Jini à comunidade, para que seja usado livremente, extendido ou e melhorado em um processo aberto similar à OMG. Jini ainda está na etapa da especificação de seus serviços, mas já existem implementações disponíveis.

4.7. Comparação e discussão dos projetos

Comparando os projetos descritos anteriormente pode-se resumir o seguinte:

- Todos os projetos apresentados neste capítulo, assim como outros, estão sendo implementados em Java pelas características de mobilidade e funcionalidades que apresenta de forma natural (segurança, RMI, JNDI, JTA, JDBC).

- Excluindo os Jumping Beans, todos os projetos apresentam um modelo "peer-to-peer", onde os agentes podem viajar entre as agências sem intermediários. No caso do Jumping Beans, a arquitetura é baseada no modelo cliente/servidor, onde o agente, para viajar até um agência, tem que passar pelo servidor.

- No referente a transporte, excluindo-se Aglets, todos eles utilizam sockets. O projeto Grasshopper utiliza, como opção, RMI.

- A única plataforma que utiliza um protocolo específico para agente é o Aglet (protocolo ATP).

- Aderem ao MASIF as plataformas Aglet, Grasshopper e MOA.

- Todos eles apresentam diferentes formas de nomear os agentes e as agências.

A análise destes projetos demonstra a factibilidade de construção de plataformas para o suporte de agentes móveis. Todos eles apresentam diferentes soluções para as mesmas questões de comunicação, segurança, transporte, execução e gerenciamento.

Também fica clara a complexidade da definição das APIs necessárias para definir todos os serviços que uma agência deveria oferecer. Ainda não está claro quais seriam todas elas.

4.8. Proposta deste trabalho

Se se quer que a tecnologia dos agentes móveis tenha sucesso, em grande escala, é necessário que as propostas sejam aplicáveis para qualquer ambiente ou para aquelas onde sua aplicabilidade seja mais evidente. Desta forma, poderá construir-se uma rede de agentes, sob o conceito do "agent grid", que seja atrativa para os usuários, no sentido que ofereça diversas alternativas, de tal forma que seja possível escolher aquela que seja mais adequada às suas necessidades, desde que se assegure a interoperabilidade e portabilidade do produto.

A tecnologia Jini parece adequada para constituir uma plataforma base para o suporte de agentes móveis. Mas ela pertence a uma empresa, o que não é vantajoso para os usuários que ficariam dependentes de uma tecnologia proprietária. Contudo poderia servir como modelo para especificação de um sistema de agentes ("agent grid") baseado em CORBA, da mesma forma como EJB tem servido para a especificação do Modelo de Componentes CORBA. Neste caso, a tecnologia Jini seria uma outra especialização que poderia coexistir com outras implementações. É claro que o processo de desenvolvimento de uma especificação baseada em CORBA é mais lento que uma especificação baseada em uma linguagem proprietária de programação como Java, mas apresentaria outras vantagens, como ter uma plataforma independente das tecnologias que se utilizem para sua implementação e melhores preços para os usuários.

A especificação MASIF só aborda a questão da interoperabilidade entre agências de diversos fabricantes. Nesta especificação, o transporte dos agentes está muito ligado às linguagens orientadas a objetos que utilizam carregadores de classes, como o Java, o que não é adequado para uma especificação padrão, uma vez que um agente é uma entidade de software que pode implementar-se com código orientado a objetos, código baseado em procedimentos ou funções e comandos "scripts", os quais apresentam diversas formas para sua manipulação.

Como resultado desta discussão propõe-se, neste trabalho, a especificação de um serviço de transporte de agentes, baseado em CORBA. Apesar de CORBA ter sido criada originalmente para suportar objetos estáticos, apresenta grandes vantagens, como especificação independente de qualquer implementação, e já que qualquer serviço pode fazer uso daqueles existentes ou de outros que venham a ser especificados. Nesta proposta, serão especificados: as interfaces, os formatos de negociação e o formato do conteúdo do receptáculo do agente, além do formato do nome para as agências e para os agentes. Todas estas questões não foram consideradas parcial ou totalmente como relevantes, nos projetos mencionados, especialmente a questão da forma de transferência de um agente de um lugar para outro. Somente o Aglet apresenta um protocolo ATP para o transporte do agente, o qual foi proposto, sem sucesso, na submissão inicial de um Serviço de Facilidade de Agentes (MAF) da OMG. O objetivo de MAF era a especificação de um serviço para o suporte dos agentes, incluindo a migração, e evoluiu para converter-se no que se conhece hoje como MASIF, que só aborda a interoperabilidade entre agências.

4.9. Resumo

No capítulo foram descritos vários projetos de agentes móveis, considerando-se os aspectos relativos às formas de abordagem das questões de nomeação e de endereçamento das agências e dos agentes, os protocolos de transporte oferecidos e as formas de empacotamento do agente e de outros dados e meta-dados, antes do seu despacho para o destino. Na descrição foram considerados os projetos D'Agents, Aglet, Grasshopper, MOA, JumpingBeans e Jini. Outros projetos existem na literatura, mas estes foram escolhidos por se considerar que são os mais representativos nas suas características.

A partir das descrições feitas, compararam-se estes projetos e analisaram-se suas vantagens e desvantagens, para finalmente se justificar a proposta de uma especificação de um serviço de transporte de agentes baseado em CORBA. Esta proposta será apresentada e desenvolvida nos capítulos seguintes.

Parte II: Desenvolvimento de um Serviço de Transporte de Agentes

5. Funcionalidades do STA

Neste capítulo, apresenta-se a proposta de uma arquitetura, para construir uma plataforma que suporte o transporte de agentes, a qual se denominará Serviço de Transporte de Agentes (STA). Ela é parte de um sistema maior, denominado Agência ou Servidor de Agentes, cuja arquitetura foi apresentada no capítulo 2 deste documento, abrangendo um conjunto de serviços que cobre os aspectos de comunicação, transporte, execução, segurança e gerenciamento dos agentes. Neste trabalho, a arquitetura proposta considera unicamente o aspecto do transporte de agentes, e que deverá satisfazer os seguintes requisitos:

- Implementar uma agência de forma distribuída por que está baseada na arquitetura CORBA. Logo, deve fazer uso dos serviços e vantagens que essa arquitetura apresenta, por exemplo:
  - Interoperabilidade entre as agências.
  - Implementação em diferentes linguagens desde que tenham especificado mapeamento com IDL.
  - As diversas funcionalidades podem-se implementar em forma distribuída, o que permitirá fornecer diversos ambientes para executar os agentes móveis.
- O serviço deverá fornecer invocações assíncronas baseadas no Serviço de Eventos;
- Para o cliente deste serviço, que pode ser um agente ou o dono do agente, três tipos de mobilidade serão suportadas: réplica, movimento e migração.
- A transferência de um agente é feita em duas etapas: negociação [Dale97] e transporte;
- O transporte pode ser feito utilizando o protocolo definido durante a fase de negociação (IIOP, TCP, SMTP, FTP, etc.);
- Os nomes das agências são registrados através do Serviço de Nomes de CORBA;
- Os nomes dos agentes são registrados através do Serviço de Registro de Agentes, o qual deve ser especificado.

É importante salientar que em cada uma das etapas no desenvolvimento deste serviço, se fez uso da linguagem UML para documentar o projeto de software. Para isso, utilizou-se o Rose98 [Rational98] como ferramenta de engenharia de software.

5.1. Nomes e localização

Um dos objetivos dos sistemas distribuídos é a transparência na localização, ou seja, que a localização física dos objetos não é visível nem importante para o cliente [Peine97]. Por outro lado, os agentes móveis caracterizam-se por movimentar-se entre posições físicas.

Como a proposta está baseada em sistemas distribuídos e orientada a objetos, isso não deveria causar problemas. Na maioria dos projetos atuais, a aplicação deve especificar a posição física dos objetos, por exemplo, utilizando formatos de nomes simbólicos, como o endereço URL, que reflete a posição física do hospedeiro ou através de um número IP, quebrando uns dos objetivos fundamentais dos sistemas distribuídos. Por exemplo, se um documento "html", tal como a página "web" de um usuário, se movimentar para outro servidor "web" ou para outro diretório, o endereço URL terá que ser modificado, ou seja, o documento "html" passará a ter outro endereço [Hauck96].

Portanto, a plataforma que suporta os agentes móveis, deverá oferecer um sistema de nomes e localização baseado só no nome dos objetos (agências, agentes), preocupando-se ela associar esses nomes à posição real dos objetos. Para a aplicação a posição física real dos objetos estáticos (agências) e móveis (agentes móveis) será transparente.

Isto leva a descrever um cenário aonde os agentes móveis vão se movimentar através das agências, sem se preocupar com o seu lugar físico ou com outros recursos de que precise, senão de saber o nome delas. Na figura 5.1, apresenta-se um cenário onde um agente está-se movimentando entre as agências "A", "B" e "C", as quais poderão ser constituídas por: um computador, por um computador de grande porte ou por uma rede de computadores. Estas agências estão instaladas sobre ORBs de fabricantes diferentes ("x", "y", e "z") e portanto, utilizam IIOP para comunicar-se entre si; na realidade, os ORBs comunicam se via IIOP, utilizando a rede global (exemplo, a Internet) e as agências através destes ORBs. O agente poderá utilizar diferentes protocolos para movimentar-se entre as agências: na figura utiliza-se primeiro o protocolo IIOP para ir até a Agência B, e posteriormente "Sockets" para viajar até a Agência C.

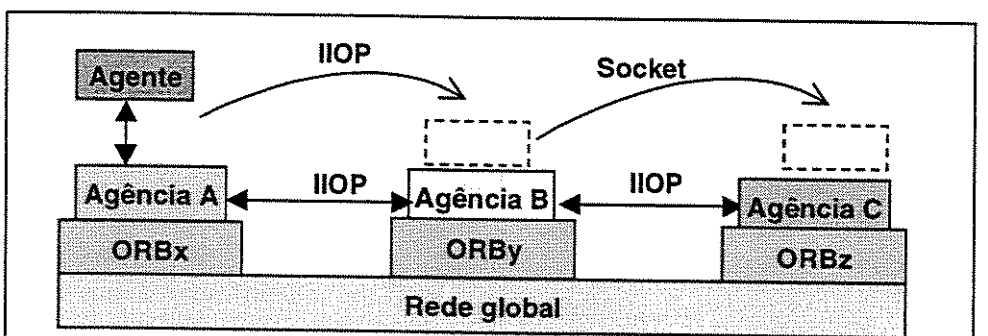

Figura 5.1: Movimentação de um agente em uma plataforma de agências heterogêneas

Cada agência aparecerá oferecendo mais ou menos facilidades para atender os agentes visitantes. Por exemplo, poderão existir agências capacitadas para executar agentes em ambientes WinNT/95, Sun Solaris e Linux, além de fornecer ambientes de execução para diversas linguagens de programação.

Se o objetivo é implementar esta plataforma em uma rede global, como a Internet, então é necessário definir formatos de nomes que sejam únicos globalmente, além de ter um

controle descentralizado. A seguir, apresenta-se o formato dos nomes que foi definido para nomear as agências e os agentes móveis. Decidiu-se que, para o "place", não é necessário definir um formato, porque ele tem um significado local. Por outro lado, a comunicação vai acontecer entre agências ou entre os agentes ou entre uma agência e um agente. Quando for solicitada a referência a um agente, com certeza estará envolvida a referência de um "place", ou seja, de forma opaca para a aplicação.

5.1.1. Os nomes de uma agência

Achou-se adequado definir um formato de nomes hierárquicos associado ao domínio Internet ao qual a agência pertence. Como tal, o domínio não deve explicitar o nome de uma máquina para que a distribuição seja transparente para os agentes.

Por outro lado, tem-se definido a agência como um objeto Corba, logo, seu nome deveria seguir a estrutura definida pela especificação do Serviço de Nomes Interoperáveis, a qual ainda está sob aprovação na OMG [IntNaming98a] [IntNaming98b]. Mas, em geral ela especifica uma versão em formato "string" para a estrutura de nomes original (Id, Kind), que se adotará. Desta forma e utilizando a nomenclatura BNF, define-se o seguinte formato:

< nome_agência > ::= < domínio_agência > "/" < string_nome>

< string_nome> ::= <string_componentes_nome>
["/"<string_componentes_nome>]*

<string_componentes_nome> ::= <Id> ["-" <Kind>]

< dominio_agência > ::= <Istring>

<Id> ::= <Istring>

<Kind> ::= <Istring>

Onde:

Domínio_agência é o nome do domínio ao qual pertence a agência;

< string_nome> é o nome que tem a agência dentro do domínio: sua estrutura está definida pelo Serviço de Nomes da OMG, ou seja, é uma versão "string" da estrutura de componentes (Id, Kind). Neste caso escolheu-se o símbolo "-" e não o ".", para separar ambos componentes (se o "Kind" é indicado) e evitar confusões com o nome do domínio que utiliza o símbolo "." como separador.

Por exemplo:

dca.fee.unicamp.br/agência-com/sucursal_A

5.1.2. Formato do nome de um agente

Se o nome de um agente é associado com o nome da agência onde ele é criado, e como o nome da agência é globalmente único, então basta acrescentar um nome adicional, que seja único na agência, para ter novamente um nome globalmente único e de controle descentralizado. Só é necessário separar ambos os componentes com um símbolo, para o qual escolheu-se o "#". Logo, o formato é o seguinte:

<nome_agente> ::= < nome_agência > "#"<Istring>

Por exemplo:

dca.fee.unicamp.br/agência-com/sucursal_A#Ag007

indicando que se trata do agente "Ag007", pertencente à agência "agência-com/sucursal_A", que está no domínio "dca.fee.unicamp.br".

No caso de se querer nomear um agente que seja réplica ou filho de outro agente, então acrescenta-se seu nome ao nome do agente-pai, separado pelo símbolo "+". Por exemplo:

dca.fee.unicamp.br/agência-com/sucursal_A#Ag007+Junior

Onde Júnior é um agente que é réplica do agente Ag007.

É claro que os símbolos "#" e "+" não podem ser partes do nome de um agente ou de uma réplica.

Pode observar-se nestes formatos que, a partir do nome de um agente, é possível saber o nome do agente-pai (se o nome é de um agente filho), e saber o nome da agência ao qual pertence o agente.

5.2. Movimentação dos agentes

O STA oferece três tipos de movimentação de agentes: migrar, mover e replicar, cujas seqüências de operações, geradas pelo serviço são mostradas na figura 5.2.

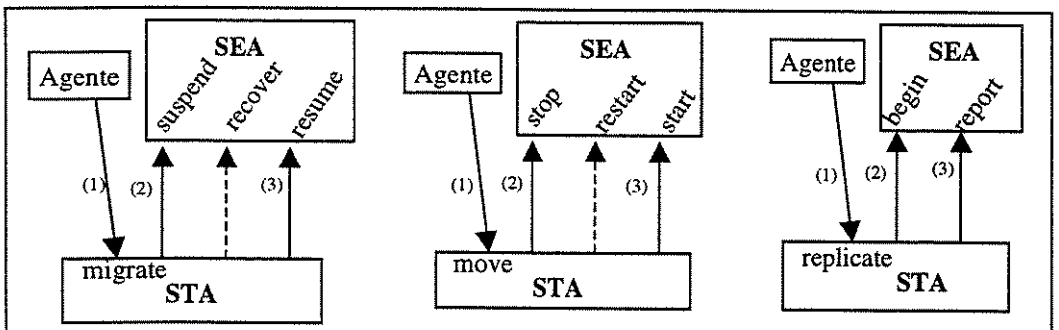

Figura 5.2: Diferentes tipos de movimentação de agentes

- **Migrar:** Neste caso o STA solicita ao Serviço de Execução de Agentes (SEA), através da operação "suspend", suspender a execução do agente, o que significa que o SEA deverá salvar o estado do agente. Posteriormente, o STA realiza internamente todas as operações necessárias para transferi-lo para o destino, onde o STA desse lugar invocará a operação "resume" do SEA, para ativá-lo, a partir do estado anterior. O STA da origem procederá a apagar o agente e seus registros. Se a transferência não tem sucesso, o STA da origem invocará a operação "recover" do SEA para reativar o agente, informando-o sobre os motivos do fracasso na transferência.

- **Mover:** O STA invoca a operação "stop" do SEA, para deter a execução do agente, sem salvar seu estado. Em seguida o STA procede a transferir o agente para o destino. Se tudo der certo, o STA do destino invocará a operação "start" do SEA para ativar o agente e o STA da origem apagará o agente e seu registro na origem. Se o processo não tem sucesso, o STA da origem invocará a operação "restart" do SEA para reativar o agente, expressando-lhe os motivos da falha.

- **Replicar:** Neste caso, o STA começará imediatamente com o processo de transferência do agente. No destino, o STA invocará a operação "begin" do SEA, para que execute o agente que chegou. Posteriormente, o STA da origem invocará a operação "report" do SEA para entregar ao agente-pai uma referência da sua réplica, se a operação for executada com sucesso, e informá-lo sobre o resultado da operação.

Nestas operações, indicar-se-ão o nome do agente e seu destino como parâmetros de entrada. Na operação de réplica, além disso, se indicar-se-á o nome do agente–filho sendo transferido como cópia do original.

5.3. Arquitetura do STA

A seguir, propõe-se uma arquitetura para o Serviço de Transporte de Agentes (STA) que cumpra com as características indicadas anteriormente. A arquitetura proposta apresenta-se na figura 5.3.

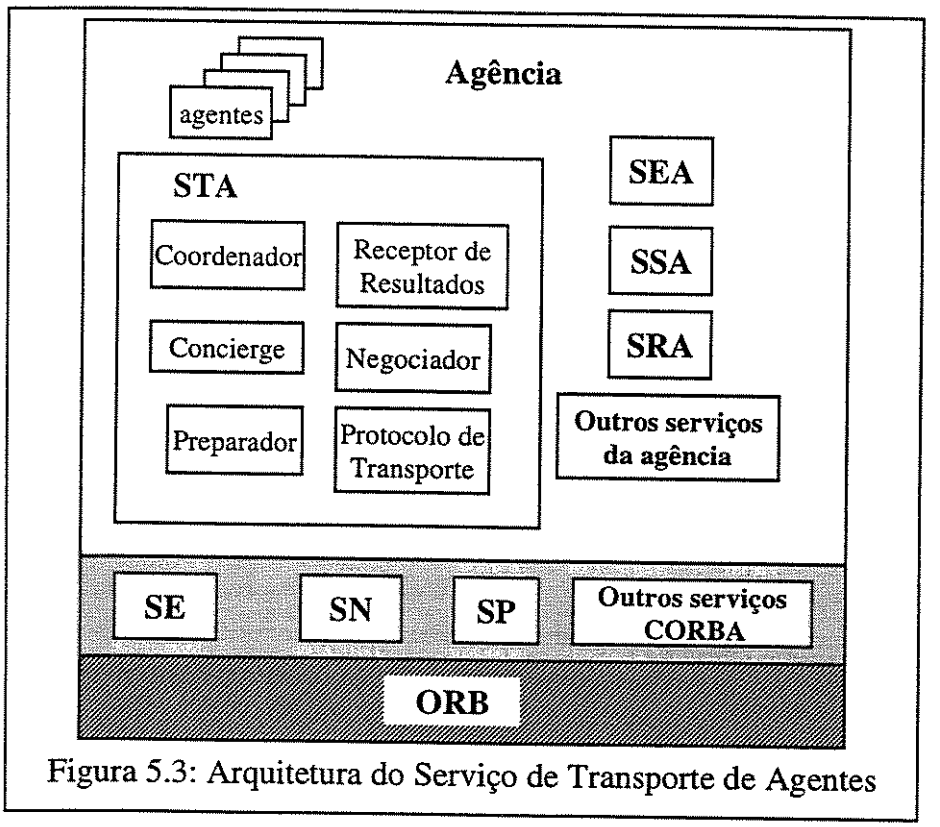

Figura 5.3: Arquitetura do Serviço de Transporte de Agentes

Para poder fornecer seus serviços, o STA precisa usar outros como: Serviço de Execução de Agentes (SEA), Serviço de Segurança dos Agentes (SSA) e Serviço de Registro de Agentes (SRA), os quais são parte da Agência. Também precisará usar os seguintes serviços CORBA já especificados pela OMG: Serviço de Evento (SE), Serviço de Nomes (SN) e Serviço de Propriedades (SP).

Para o transporte, o agente é considerado como um conjunto de componentes de software (código, dados, meta-dados) colocados em arquivos sob uma estrutura de diretórios geral, como se indicará mais para frente no documento.

Na figura 5.4, apresenta-se a relação entre os diversos componentes do STA e de outros componentes de uma agência. A seguir, explica-se cada um destes módulos funcionais que compõem a arquitetura, e a sua relação com outros serviços.

5.3.1. Coordenação

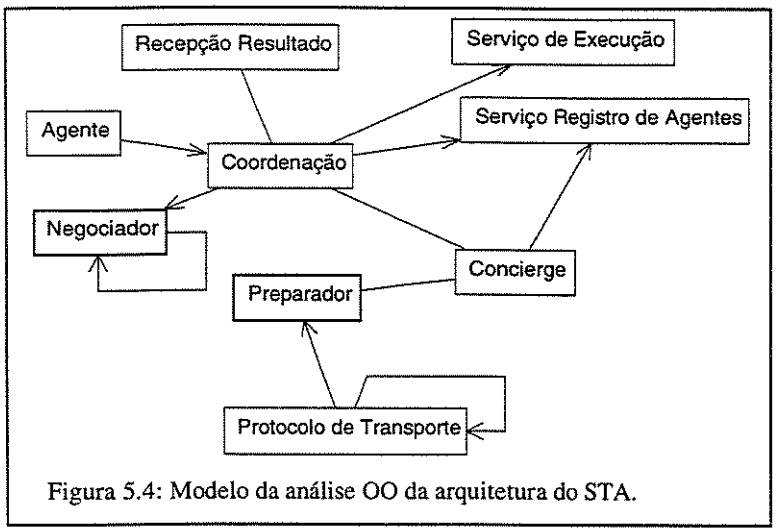

Figura 5.4: Modelo da análise OO da arquitetura do STA.

Este módulo tem como função coordenar os principais passos do STA. Isto é feito quando:

(a) Uma operação de movimento (réplica, movimento ou migração de um agente) é invocada pelo agente ou pelo seu dono;

(b) quando um agente chega à agência, estando pronto para ser executado e;

(c) quando se receba o resultado de uma operação de movimentação de um agente para outra agência.

Por exemplo, quando um agente invoca uma operação de migração indicando seu nome e o nome da agência que quer visitar, a Coordenação solicita primeiro ao Serviço de Execução suspendê-lo, para posteriormente iniciar o processo de negociação com a agência destino. Se o agente é aceito pelo destino, instrui o Concierge para proceder a despachá-lo. No caso da operação "replicate", o Coordenador primeiro registra o nome do agente-filho dentro do registro do agente-pai, junto com o nome da agência de destino: na seqüência o processamento é similar ao descrito no caso anterior.

Por outro lado, quando um agente chegar e estiver pronto para ser executado, o Coordenador é avisado pelo Concierge deste fato e solicita ao Serviço de Execução para executar o agente. Posteriormente, informa à origem, através do Receptor de Resultados, que o agente foi executado com sucesso (ou caso negativo, qual a causa).

Assim, quando o Coordenador receber do Receptor de Resultados que a operação de movimentação de um agente terminou com ou sem sucesso, deverá determinar o que fazer para fechar corretamente o serviço. Para isso, consulta o registro do agente, através do SRA, para saber qual tipo de movimentação estava em curso e ordena ao Receptor de Resultados a realização da seqüência de operações para finalizar o serviço.

5.3.2. Negociação

Este módulo atende as solicitações do Coordenador, quando este último tenha que negociar a transferência de um agente para uma agência-destino. Dois aspectos serão negociados:

(a) a possibilidade de instalar e executar o agente e

(b) a definição do protocolo de transporte para transferir o agente.

A negociação e instalação consistem em enviar informação, por exemplo, sobre o sistema operacional, espaço que ocupa no disco, memória mínima de execução, tipo da linguagem de programação, etc. Portanto, haverá que padronizar um formato de negociação, o qual deverá ser suficientemente flexível para futuras modificações. Para implementar esta etapa, duas possibilidades foram analisadas: enviar os valores de negociação como parâmetros da invocação ou colocá-los dentro de um arquivo de texto e este último enviá-lo como arquivo binário, para ser depois lido na recepção. A primeira alternativa pode especificar-se da seguinte forma:

```
typedef string PropertyName;
typedef string PropertyValue;

struct Property {
      PropertyName name;
      PropertyValue value;
};

typedef sequence<Property> PropertySeq;
typedef PropertySeq ApplicationProfile;

struct Protocol {
  TypeProtocol protocol_name;  // Protocolo escolhido
  PropertySeq detail_tx_prot;  // Informação relativa ao protocolo escolhido
};

interface Negotiate {
 boolean booking(in ApplicationProfile item_list,
                 out Protocol response);
};
```

Contudo apresenta um formato pouco flexível, onde só se pode indicar o nome de uma propriedade e seu valor (ou um conjunto de valores separados por um caractere especial).

Além disso, deve-se definir um padrão para o formato das propriedades: se a quantidade de propriedades a serem negociadas aumenta, significa que haverá uma sobrecarga maior na preparação (marshalling) e transmissão destes parâmetros, tornando mais lento o despacho dos mesmos.

A segunda alternativa permite definir um formato mais flexível, utilizando o padrão XML [Xml97], que já é do domínio público, e sem necessidade de criar novos padrões. XML permitirá a troca de dados sem perdas de significado entre sistemas heterogêneos e a transmissão dos mesmos como um arquivo binário. Um arquivo deste tipo será enviado mais rapidamente através de uma invocação CORBA, porque não se precisa do trâmite de preparação dos dados.

Por estes motivos foi escolhida a segunda alternativa. Este arquivo, que foi nomeado "negotia.tor", é parte dos componentes do agente e poderá ser lido por qualquer agência que adira a este padrão.

Inicialmente, considerou-se importante negociar as seguintes propriedades de um agente:

- Tipo de movimentação que está realizando o agente;
- Seu tamanho, ou seja, a soma total dos arquivos que compõem o agente;
- Espaço de disco disponível que a agente precisa para fazer seu trabalho no local;
- A memória ativa mínima de que precisa para a execução;
- A linguagem na qual foi escrito o agente;
- O compilador de que precisa, no caso de ser compilável;
- Sistemas operacionais que suportam o agente.

Também, inclui-se o nome do agente para que seja identificado pelo destino. A estrutura deste documento de negociação deve ficar definida em XML, em um arquivo do tipo DTD ("Document Type Declaration"), interpretável em qualquer lugar em que chegue. Este arquivo, nomeado como "Negotiate.dtd", ficará alojado na pasta nomeada "meta-info" da pasta de trabalho do agente, tendo a seguinte estrutura:

```
<!-- Document Type Declaration para a negociação de um agente      -->
<!-- Data: 18/05/99                                                -->

<!ELEMENT NEGOTIATION INFO>

<!ELEMENT INFO (AGENT_NAME, KIND_OF_MOVING, AGENT_SIZE,
```

```
DISK_SPACE, MEMORY_NEEDED, LANGUAGE, COMPILER*,
OPERATING_SYSTEMS)>

<!ELEMENT AGENT_NAME (#PCDATA)>

<!ELEMENT KIND_OF_MOVING ("migration" | "moving" | "replication" )    >

<!ELEMENT AGENT_SIZE (#PCDATA)>
<!ATTLIST AGENT_SIZE unity (K|M|G) "K">

<!ELEMENT DISK_SPACE (#PCDATA)>
<!ATTLIST DISK_SPACE unity (K|M|G) "K">

<!ELEMENT MEMORY_NEEDED (#PCDATA)>
<!ATTLIST MEMORY_NEEDED unity (K|M|G) "K">

<!ELEMENT LANGUAGE (NAME, VERSION) >

<!ELEMENT COMPILER (NAME, VERSION)  >

<!ELEMENT OPERATING_SYSTEMS OS+>

<!ELEMENT OS (NAME, VERSION)>

<!ELEMENT NAME  (#PCDATA)>
<!ELEMENT VERSION (#PCDATA)>
```

Estas declarações definem os seguintes elementos:

AGENT_NAME: Especifica o nome do agente como do tipo PC#DATA, ou seja, é uma seqüência de caracteres. Este elemento é obrigatório.

KIND_OF_MOVING: Especifica que tipo de movimentação está realizando o agente. Pode ter um dos seguintes valores: "migration", "moving" ou "replication" . Este elemento é obrigatório.

AGENT_SIZE: Especifica o tamanho que tem o agente. Ou seja, a soma total de seus componentes que vai movimentar-se. O valor pode se expresso em unidades de Kbytes, Mbytes ou Gbytes, o qual é definido pelo atributo "unity" atribuído no ATTLIST do elemento. Se a unidade não é indicada, então se assume o Kilobyte como "default". Este elemento é obrigatório.

DISK_SPACE: É especificado, de forma similar ao anterior, como espaço livre mínimo em disco para que o agente realize seu trabalho. Este elemento é obrigatório.

MEMORY_NEEDED: É especificado de forma similar ao anterior, como espaço mínimo de memória ativa que o agente necessita para sua execução. Este elemento é obrigatório.

LANGUAGE: Especifica a linguagem em que está programado o agente. Para isso, são especificados o nome da linguagem (elemento NAME) e sua versão (elemento VERSION). Este elemento é obrigatório.

COMPILER (NAME, VERSION): Especifica o tipo de compilador que deve ser utilizado para compilar o programa do agente, no caso que esteja programado em uma linguagem compilável. É especificado como um elemento que pode estar ausente na negociação, portanto, é opcional.

OPERATING_SYSTEMS: Indica os sistemas operacionais nos quais o agente pode executar-se. Cada sistema operacional é especificado pelo elemento OS+: o símbolo "+" indica que pode ser expressado mais de um sistema operacional, cada um deles, pelos elementos NAME e VERSION, que expressam o nome e versão do sistema operacional. Este elemento é obrigatório.

Como exemplo, apresenta-se um documento que negocia estas propriedades, baseada na estrutura anteriormente definida:

```
<?xml version= "1.0" ?>

<!DOCTYPE NEGOTIATION SYSTEM "Negotiate.dtd">

<!-- O anterior está informando que o arquivo nomeado "Negotiate.dtd" contém -->
<!-- a estrutura definida para este documento. A estrutura corresponde a um    -->
<!-- documento do tipo NEGOTIATION                                              -->
```

```
<NEGOTIATION>

    <TITLE> Negociação de um agente </TITLE>

    <AGENT_NAME>dca.fee.unicamp.br/Agência1#Ag001</AGENT_NAME>

    <KIND_OF_MOVING>migration</KIND_OF_MOVING>

    <AGENT_SIZE unity= "K">3</AGENT_SIZE>

    <DISK_SPACE unity= "M">1</DISK_SPACE>

    <MEMORY_NEEDED unity= "K">10</MEMORY_NEEDED>

    <LANGUAGE>
        <NAME>Java</NAME>
        <VERSION>1.2</VERSION>
    </LANGUAGE>

    <OPERATING_SYSTEMS>
        <OS>
            <NAME>SunSolaris</NAME>
            <VERSION>1.2</VERSION>
        </OS>
        <OS>
            <NAME>Linux</NAME>
            <VERSION>2.1</VERSION>
        </OS>
    </OPERATING_SYSTEMS>

</NEGOTIATION>
```

Este documento especifica que:

- Trata-se de um agente em migração nomeado: "dca.fee.unicamp.br/Agência1 #Ag001";
- Tem um tamanho de 3Kbyte;
- Precisa de 1Mbytes de espaço no disco e de 10Kbytes de memória ativa;
- Está escrito na linguagem Java, versão 1.2, para as plataformas SunSolaris versão 1.2 e Linux versão 2.1.

Para negociar o protocolo de transporte, a agência local lerá, a partir do arquivo "locomot.ion" do agente, que está escrito em ASCII, a lista dos protocolos de transporte que deseja utilizar, na ordem especificada. Se algum dos protocolos especificados pelo agente não for suportado pela agência, então será eliminado da lista que finalmente será enviada à agência destino para sua negociação. A lista de protocolos de transporte proposto será enviada como uma seqüência de parâmetros dentro da mesma invocação anterior.

Por outro lado, o negociador do destino atenderá a solicitações do negociador da origem para determinar se aceita ou não o agente. Para isso, pegará a informação recebida, procederá internamente a determinar se é factível a instalação e execução do agente e se concorda em usar um dos protocolos de transporte proposto. Se a resposta for positiva, indicará à origem o protocolo de transporte escolhido junto com os parâmetros necessários para sua utilização. Além disso, deverá fazer a reserva para o agente, para que quando o agente chegue, o destino verifique se tem reserva feita e aceite sua instalação.

5.3.3. Concierge

Este módulo tem como funções atender as solicitações do:

(a) Coordenador, quando um agente vai ser despachado.
(b) Preparador, quando é consultado para saber se o agente tem reserva feita.
(c) Preparador, quando é avisado que um agente chegou e já está instalado.

Quando o agente vai ser despachado, o Concierge solicita ao Preparador que proceda a preparar o agente para ser transportado Isto significa que vai ser compactado e posteriormente empacotado sob um formato especifico.

Quando é consultado pelo Preparador sobre se o agente tem reserva pronta, o Concierge consultará o Serviço de Registro de Agentes.

Quando é informado pelo Preparador que um agente chegou e está instalado, o Concierge avisa o Coordenador que um agente está pronto para ser executado.

5.3.4. Preparação

Este módulo atende as solicitações do:

(a) Concierge, quando um agente precisa ser preparado (compactado e codificado).

(b) Protocolo de Transporte, em duas oportunidades:

- Primeiro, quando é necessário obter o nome do agente a partir do pacote que está sendo recebido, para que consulte o Concierge sobre a reserva do mesmo.
- Segundo, quando o pacote de um agente é aceito e deve ser definitivamente decodificado, descompactado e instalado, avisando posteriormente o Concierge que um agente chegou e está instalado.

Posteriormente no trabalho será tratado em detalhe como é feita a preparação do agente para sua transferência e como é feito o processo inverso quando um agente chega ao destino.

5.3.5. Protocolo de Transporte

Este módulo atende as solicitações do Concierge para transferir um agente, que indica o destino e o protocolo que deverá usar.

No destino, este módulo deverá fornecer os mecanismos para permitir a recepção do agente, sob o protocolo de transporte escolhido pelas agências que participam na movimentação do agente. Quando começa a receber o pacote, imediatamente solicita ao módulo de Preparação, que obtenha o nome do agente e determina se tem reserva feita: caso positivo continua com a recepção do resto do pacote. No término da recepção, entrega o controle do processo ao Preparador para fazer o desempacotamento, descompactamento e instalação do agente.

Portanto, este módulo suporta os protocolos de transporte que os agentes poderão usar para seu transporte entre as agências. Como já foi explicado anteriormente, durante a negociação a agência origem obtém do destino os parâmetros necessários para usar o protocolo corretamente.

Como a arquitetura das agências está baseada no CORBA, o protocolo IIOP será o protocolo de transporte básico em todas as agências. Outros protocolos poderão ser usados como alternativa ao anterior.

5.3.6. Recepção do Resultado

Este módulo tem como função receber do destino, informação sobre o resultado da operação de movimentação do agente, o qual é repassado ao Coordenador da agência através do serviço de eventos, para que tome as medidas necessárias e também, para ajudar o Coordenador a resolver sobre como terminar a operação de movimentação do agente. Por exemplo, se a movimentação de migrar ou mover tem sucesso, o registro e os componentes do agente serão apagados: se não tiver sucesso, o Coordenador invocará a operação adequada na Unidade de Execução, para que ela reative o agente e informe ao mesmo do resultado negativo da movimentação (por exemplo: a operação "recover" para a migração e "restart" para mover). Além disso, restabelece, dentro do registro do agente, a situação do mesmo como "pronto".

Se o resultado da operação se refere a uma réplica com sucesso, então este módulo recebe uma referência para o agente-filho que foi ativado no destino, a qual é repassada ao Coordenador. No caso das réplicas, o agente que invocou a operação (agente-pai), não vai ser apagado e só seu registro será atualizado para levar em conta a situação dos agentes filhos.

5.4. Resumo

Neste capítulo analisaram se as funcionalidades que um serviço de transporte de agentes deverá apresentar, definindo-se três tipos de movimentação para os agentes: migrar, mover e replicar. Também, definiram-se os formatos para os nomes das agências e dos agentes para que sejam únicos globalmente.

Baseada nas funcionalidades e na plataforma CORBA, propõe-se uma arquitetura para um Serviço de Transporte de Agentes, explicando-se cada módulo que a compõe e a relação entre eles.

Este será um serviço que permitirá a uma aplicação saber se pode enviar o agente ao destino e qual protocolo usar para seu transporte, através de negociação com a agência destino, e utilizando XML para definir os termos que serão usados na negociação. Desta forma, espera-se que o agente só seja transferido se realmente for aceito.

O serviço será assíncrono para que a aplicação tenha a opção de continuar com seu trabalho, uma vez que o processo de movimentação do agente poderá ser longo.

É importante salientar que este serviço movimentará qualquer tipo de agente. Na realidade, pegará a área onde se encontram todos os arquivos do agente, comprimirá esses em formato *zip*, gerando um pacote que será transferido ao destino.

Finalmente, na arquitetura estabelece-se que várias das funcionalidades deste serviço utilizarão serviços CORBA já existentes, como os serviços de Nomes e de Eventos. Destaca-se a necessidade de considerar novos serviços, cujas funcionalidades ficam fora do STA, como o Serviço de Registro de Agentes.

No capítulo seguinte especifica-se este serviço.

6. Operação e especificação do STA

Continuando com a apresentação da proposta, neste capítulo se descreverá como opera o STA e como foi feita a especificação, em IDL, das interfaces públicas que este serviço terá para ser implementado em qualquer ambiente e linguagem de programação (desde que tenha definido o mapeamento). Também, incluiu-se a descrição do formato do pacote, onde o agente deve ser colocado para seu transporte até ao destino, e a descrição dos componentes do agente, como da estrutura de diretório básica de seu ambiente onde ele é instalado. Assim, a agência poderá manipular e gerenciar qualquer agente que receba.

6.1. Operação do STA

Para esclarecer ainda mais a forma como este serviço opera, nas Figuras 6.1 e 6.2 respectivamente apresentam-se diagramas de colaboração para descrever dois cenários: despacho de um agente com negociação e recepção de um agente, ambos com sucesso, os quais correspondem à migração como um caso de uso do STA.

6.1.1. Despacho de um agente

Neste cenário um agente decide migrar invocando (1) a operação *migrate*. Logo, o coordenador local procede (2) a suspender o agente, (3) ordena que a transferência do

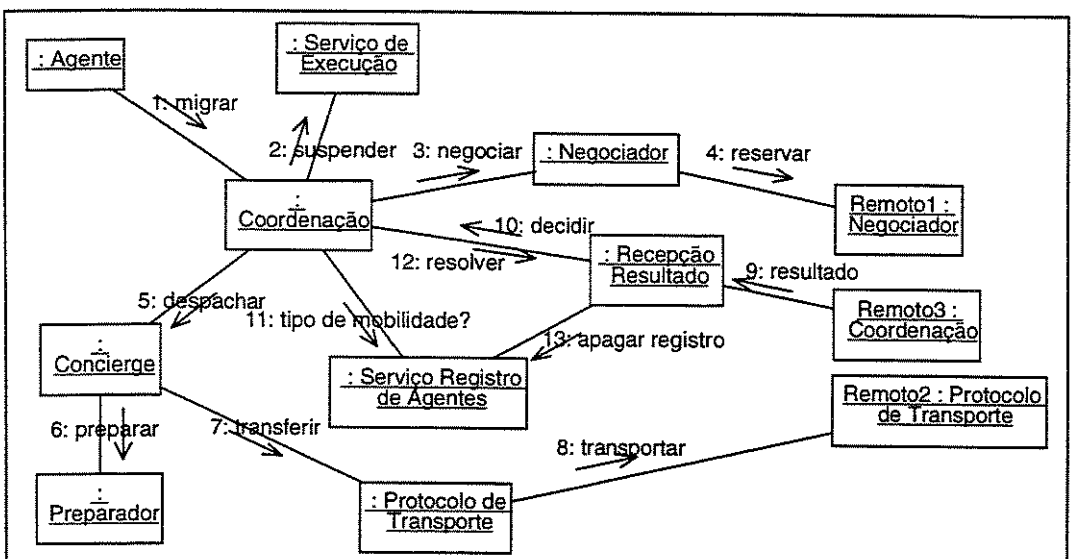

Figura 6.1: Diagrama de Colaboração para o cenário de Despacho de um agente na migração

agente seja negociada, deixando uma vaga reservada no destino se a negociação tem sucesso. O negociador local (4) negocia com seu homólogo remoto, se aceita o agente e qual protocolo de transporte usar; nesta seqüência, o negociador remoto vai reservar uma vaga para o agente, registrando: seu nome junto com o nome do diretório de trabalho que

lhe foi consignado, sua situação como "reservado", tipo de movimentação como "migração" e a data atual. Como neste cenário a negociação tem sucesso, a Coordenação local ordena ao Concierge (5) despachar o agente, que, por sua vez, ordena ao Preparador (6) que comprima os componentes do agente e prepare o pacote para ser enviado. Posteriormente o Concierge ordena (7) ao Protocolo de Transporte que transfira o pacote para o destino indicado, usando o protocolo definido durante a negociação, que (8) transporta o pacote até o destino. No lado remoto, o agente é instalado e executado com sucesso, e a Coordenação remota envia (9) ao Receptor de

Resultado local o resultado da operação. No lado local novamente, o Receptor de Resultado (10) informa à Coordenação que o agente foi transferido com sucesso, ficando à espera de uma decisão para terminar o processo de migração. A Coordenação (11) consulta o Serviço de Registro de Agentes, sobre o tipo de mobilidade que o agente tinha (neste caso migração). Com essa informação, a Coordenação ordena ao Receptor de Resultado (12) resolver, quem procederá a apagar os componentes do agente, incluindo seu diretório de trabalho e (13) seu registro através do Serviço de Registro de Agentes.

6.1.2. Recepção de um agente

Neste cenário assume-se que, produto de uma negociação com sucesso, foi feita a reserva de um agente. A seqüência inicia-se quando o Protocolo de Transporte local começa a (1)

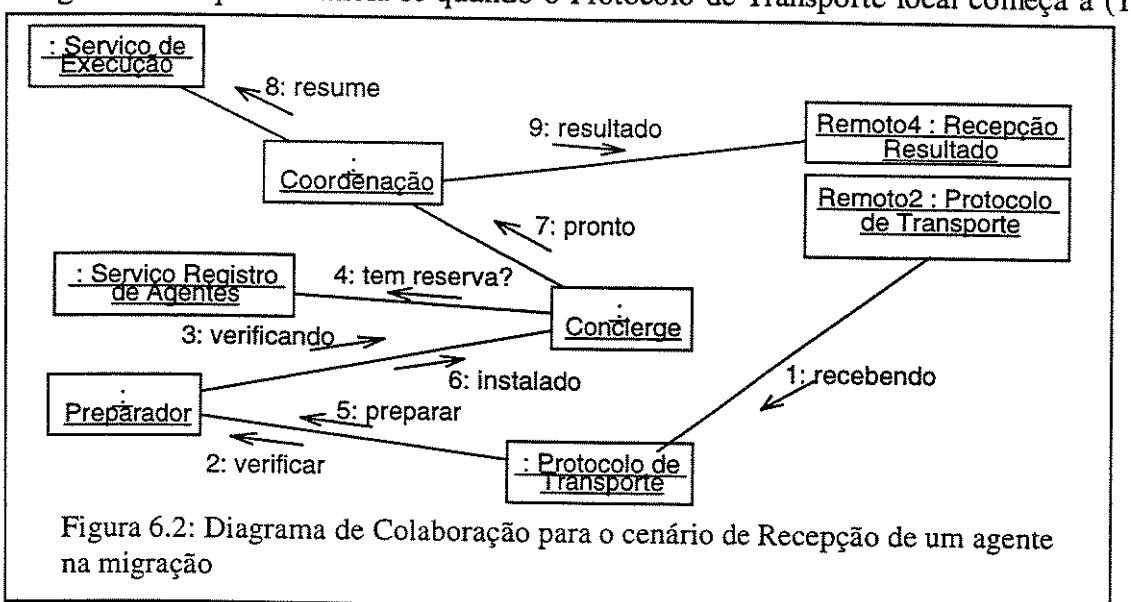

Figura 6.2: Diagrama de Colaboração para o cenário de Recepção de um agente na migração

receber o pacote de um agente, que imediatamente solicita ao Preparador que (2) verifique se o pacote que está recebendo corresponde a um agente com reserva no local. Para isso o Preparador decodifica o pacote só para ler o nome do agente e solicita ao Concierge que (3) verifique se o agente indicado tem reserva. O Concierge, através do Serviço de Registro de Agente (4) verifica se o agente tem reserva. Como neste cenário o agente tem reserva pronta, então o Concierge recebe do Serviço de Registro de Agentes, além da confirmação positiva da reserva, o diretório de trabalho que foi consignado ao agente, durante a negociação. Estes dados são retornados ao Preparador para que saiba onde instalar o agente. Terminado o processo de verificação, o Protocolo de Transporte local continua recebendo o

pacote do agente, e uma vez que foi recebido totalmente, solicita ao Preparador (5) que prepare o agente para instalá-lo. O Preparador procede a decodificar e descomprimir o agente, instalando-o no diretório de trabalho correspondente, informando ao Concierge que (6) um agente foi instalado. O Concierge (7) avisa à Coordenação que o agente indicado está pronto para ser executado. Para isso, a Coordenação solicita ao Serviço de Execução que reinicie a execução do agente (*resume*), avisando posteriormente à Recepção de Resultado remota, (9) que o agente indicado foi executado com sucesso no novo destino, finalizando assim o processo de migração com sucesso do agente.

6.2. Codificação e empacotamento do agente

Como já foi explicado anteriormente, o módulo de Preparação é o encarregado de tratar o agente antes da sua transmissão e depois de ter sido recebido pelo módulo do Protocolo de Transporte.

O tratamento consiste na compactação dos componentes do agente e na codificação de um pacote que inclui o agente compactado, mais dados de controle relacionados com sua codificação. Depois, no destino, o Preparador procederá a decodificar o pacote, baseado nos dados de controle e posteriormente descomprimirá os componentes do agente, instalando-o no subdiretório de trabalho definido pela agência destino.

6.2.1. Compactação do agente

O motivo principal de compactação do agente é que o tamanho dos componentes do agente decresce e, portanto, o tempo de transmissão vai diminuir, melhorando o desempenho do processo de transporte do agente. Os arquivos JAR do Java utilizam este esquema pelo mesmo motivo.

O formato escolhido para comprimir o agente é o formato Zip [Nelson97], porque é aberto e existe uma variedade de produtos disponíveis no mercado, de domínio público. Mas outros métodos de compactação podem ser escolhidos, os quais devem ser indicados na codificação do pacote. Na referencia [Méndez97], está disponível o código fonte de um programa escrito em Java, o qual faz uso do pacote java.util.zip para comprimir e descomprimir os arquivos de um diretório e seus subdiretórios. Este programa foi desenvolvido durante a realização deste trabalho, para implementar a compressão e descompressão dos componentes de um agente.

Para saber que arquivos comprimir, o Preparador procura pelo arquivo denominado "manifiest.acd", alojado no subdiretório ".../meta-info/", para ler a lista dos arquivos que compõem o agente e enviá-los ao destino.

No destino e depois que o pacote tenha sido decodificado, o Preparador descompactará o agente, utilizando a informação contida no arquivo compactado de acordo com o formato *Zip*. Em seguida os componentes do agente serão instalados a partir do subdiretório de trabalho definido pela agência.

6.2.2. Codificação do pacote receptáculo (*container*) do agente

Para transportar os componentes do agente, depois de ter sido compactado, vai-se preparar um pacote, que além de conter o agente, vai ter vários campos contendo informação de controle, para que na recepção o pacote seja identificado e decodificado corretamente.

Como informação relevante para a agência, achou-se necessário incluir só: o nome do agente, a versão do formato do receptáculo do agente e o método de compactação utilizado para comprimir os componentes do agente. Assim, quando o pacote está sendo recebido, será possível saber se o agente tem reserva no local, saber o formato e o método de compactação. No futuro, espera-se incluir outras informações relevantes como, por exemplo, as credenciais do agente e sua assinatura.

Para codificar essa informação, adotou-se como referência a sintaxe de transferência CDR (Common Data Representation) da OMG, definindo-se os seguintes campos, como se mostra na figura 6.3:

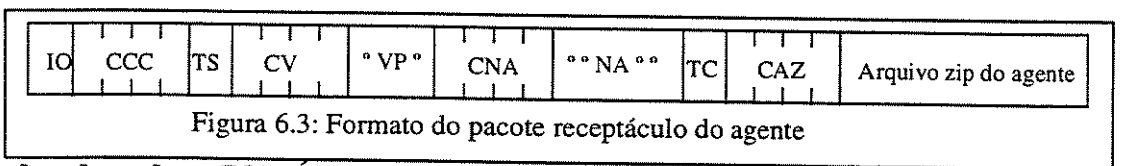

Figura 6.3: Formato do pacote receptáculo do agente

Indicador de ordem (IO): É um *byte* que indica qual ordem de *byte* está-se utilizando (0 para *Big-endian* e 1 para *Little-endian*), na representação da informação de controle de mais de um *byte* e na representação dos caracteres do nome do agente, quando é de dois ou mais *bytes*.

Comprimento do cabeçalho de controle (CCC): É um campo de 4 *bytes* para conter um número do tipo *unsigned long*, o qual indica o número de *bytes* do campo de controle. O valor não inclui os 5 *bytes* ocupados pelo primeiro campo e este.

Tipo de *string* (TS): Este é um campo de 1 *byte*, indicando se os caracteres da *string* com o nome do agente e da *string* com o nome da versão do formato, estão representados por um *byte* (valor 0) ou 2 *bytes* (valor 1).

Comprimento do campo versão do protocolo (CV): É um campo de 4 *bytes* para conter um número do tipo *unsigned long*, o qual representa o comprimento do campo de versão do protocolo.

Versão do protocolo (VP): É um campo de comprimento variável, o qual contém o nome da versão do formato do receptáculo do agente que se está usando.

Comprimento do nome do agente (CNA): É um campo de 4 *bytes* para conter um número do tipo *unsigned long*, o qual representa o comprimento do nome do agente em otetos.

Nome do agente (NA): Este é um campo de comprimento variável, o qual contém o nome do agente representado por uma *string*.

Tipo de Compactação (TC): É um campo de 1 *byte* que especifica o tipo de compactação que se está utilizando para comprimir o agente. Por exemplo: 0 para o tipo *zip*.

Comprimento do arquivo *Zip* (CAZ) : É um campo de 4 *bytes* para conter um número do tipo *unsigned long*, o qual representa o comprimento, em octetos, do arquivo que contém os componentes compactados do agente.

6.2.3. Estrutura dos componentes do agente

Como já foi explicado anteriormente a etapa de negociação está baseada no uso de informação fornecida pelo próprio agente. Para que essa informação possa ser lida nos diferentes ambientes em que o agente vai viajar, sem perder a semântica, é necessário que esteja escrita em um formato padrão. Para isso, foi escolhido o padrão XML por ser de uso universal na Web e de fácil manipulação. A informação será enviada como parte dos componentes do agente, durante a transferência do mesmo e na forma de um parâmetro binário (seqüência de octetos) durante a negociação.

Esta informação encontra-se em dois arquivos: o arquivo com a informação sobre os requisitos para a instalação e execução do agente, o qual será denominado: "negotia.tor", e o arquivo que contém o resto da informação considerada como propriedades do agente, que será denominado: "propert.ies". Na realidade, a informação contida no arquivo "negotia.tor" também forma parte das propriedades do agente. Para facilitar a negociação, foi separada do resto das propriedades em um arquivo distinto, desta forma evitando-se transferir para o destino todo o arquivo de propriedades durante a fase de negociação.

Dois arquivos ASCII serão necessários para armazenar informação sobre o agente, que, por sua simplicidade, não exigem um formato XML. Estes são a lista de protocolos de transporte que o agente deseja utilizar na ordem definida, que será denominado: "locomot.ion" e, a lista dos componentes (arquivos) do agente que precisam ser compactados. Neste caso serão registrados os nomes dos arquivos que compõem o agente, considerando a trajetória dos diretórios relativos ao diretório de trabalho que foi atribuído pela agência local: essa lista será armazenada em um arquivo que será denominado "manifiest.acd" (acd: agent component descriptor) de forma similar ao arquivo de manifesto do JavaBeans. O conteúdo destes arquivos também poderia ser considerado como propriedades, mas como são simples e de formato conhecido, não precisam de codificação em XML.

No caso do arquivo "locomot.ion", seu conteúdo serão números atribuídos a cada tipo de protocolo, por exemplo: 0, 1, 2 para os protocolos IIOP, FTP, SMTP respetivamente. Estes números serão escritos como caracteres ASCII.

No caso do arquivo "manifiest.acd", o caracter separador de subdiretórios, dentro do *string* do nome de arquivo, será o símbolo "/". Logo, cada ambiente se preocupará de manipular corretamente os nomes dos arquivos que compõem o agente, para sua compactação e descompactação. No caso do ambiente UNIX, o separador é "/" e no caso de Win95/98/NT, o separador é "\".

Todos estes arquivos que, de alguma maneira, descrevem um agente, constituindo de fato um conjunto de meta-dados, serão instalados em um subdiretório especial do diretório de trabalho, que será denominado "meta-info". Outros arquivos contendo meta-dados sobre o agente, poderão ser instalados neste diretório.

Portanto, a estrutura geral de diretórios para o agente em uma agência será a seguinte:

Diretório local/Diretório de Trabalho/meta-info/negotia.tor

Diretório local/Diretório de Trabalho/meta-info/locomot.ion

Diretório local/Diretório de Trabalho/meta-info/ manifiest.acd

Diretório local/Diretório de Trabalho/meta-info/ propert.ies

Diretório local/Diretório de Trabalho/meta-info/ outros arquivos meta-dados

Diretório local/Diretório de Trabalho/outros dir com os componentes do agente

O **Diretório local** é o diretório onde a agência instalará todos os agentes que receba: o nome dele será uma questão local.

O **Diretório de Trabalho** é um diretório que a agência define e atribui a um agente. O nome é único para diferenciá-lo dos outros diretórios de trabalho na agência, sendo salvo no registro do agente. É questão da agência como vai gerenciar estes sub-diretórios.

6.3. Serviços utilizados pelo STA

O STA utilizará diversos serviços CORBA, basicamente os serviços de: Nomes e Eventos. O serviço de Propriedades poderá ser usado no caso que as propriedades do agente queiram ser manipuladas pela agência: neste caso, o serviço de Propriedades deverá acessar os arquivos em formato XML.

Além dos anteriores, o STA deverá usar um serviço que permita registrar e gerenciar o agente. Como é específico deste ambiente, será necessário definí-lo, pelo menos, os aspectos que sejam necessários para o STA. Outras facilidades poderão ser incorporadas, na medida que os outros serviços da agência o precisem.

6.3.1. Serviço de Registro de Agentes (SRA)

Este serviço vai ser utilizado pelos outros serviços que conformam a agência para criar, modificar, consultar e apagar os registros dos agentes e também para consultar qualquer outra informação sobre o agente que não necessariamente esteja contida nos registros (por exemplo solicitação de referência a um agente).

No caso do STA, este serviço monitora o agente durante o seu transporte, nas seguintes situações:

- Quando como resultado de uma negociação, um agente é aceito e um registro para ele for criado no destino, registrando-se:

    1. O nome do agente, junto com um nome de diretório de trabalho que será criado nesse momento.
    2. A situação do agente como "reservado", junto com a data da reserva e o nome da agência origem, para que possa reenviar alguma mensagem quando aconteça algum problema com a reserva do agente ou quando precise informar a origem, sobre algum evento importante relacionado com o agente (por ex., que o agente foi executado com sucesso).
    3. Tipo de movimentação que está realizando o agente (migrar, mover ou replicar). Assim, quando a agência receber informação sobre um agente, poderá atuar de forma correspondente.

- Quando um agente está sendo despachado para o destino, a situação do agente na agência origem será de "suspenso".

- Quando um agente é recebido, ele será instalado no diretório de trabalho que foi criado durante a etapa de negociação, e a situação do agente passara ser de "pronto".

- Quando o agente é executado, a situação dele será registrada como em "execução".

- Toda vez que a situação do agente muda de valor, a data também mudará.

- Registrar os agentes-filhos, quando um agente vai-se replicar: isto significa que o nome do agente filho e seu destino serão registrados e relacionados com seu agente-pai.

Portanto, a idéia é que este serviço mantenha na agência, uma base de dados com o mínimo de informação necessária para a manipulação do agente e assim, evitar redundância na informação, o que seria negativo para o desempenho, considerando que poderá haver uma grande quantidade de agentes nas agências. O resto dos dados sobre o agente, como: data de criação, tamanho, espaço de disco necessário para sua execução, etc., que serão explicados com maior detalhe no item sobre protocolo de negociação, serão parte dos arquivos que conformam as propriedades de um agente e que por sua vez, poderão ser acessados opcionalmente através do serviço de propriedades.

É importante salientar que este serviço cumpre a função de servidor de nomes de agentes, sendo acessado através da agência que hospeda os agentes. Desta forma, os Serviços de Nomes (de objetos CORBA) não serão sobrecarregados com uma grande quantidade de nomes de agentes que normalmente existirão na rede e que estarão temporariamente nas agências, mudando de local constantemente. Além disso, permitirá a manipulação de agentes que podem ou não ser objetos CORBA, de forma semelhante ao MASIF.

Se alguma aplicação quer saber onde está seu agente, primeiro deverá contatar-se com as agências-destino (do itinerário da viagem) através do Serviço de Nomes, para posteriormente acessar o SRA das agências e consultar sobre a presença do agente. Assim, espera-se evitar o tráfego na rede global na consulta pelo paradeiro de um agente.

Em resumo, o registro do agente terá os seguintes campos:

< nome do agente> ::= <nome do agente em formato *string*>

< diretório de trabalho> ::= nome do diretório (incluindo a trajetória), onde os componentes do agente estão armazenados.

< situação do agente> ::= "reservado" | "suspenso" | "pronto" | "execução"

< data de modificação da situação do agente> ::= <dd/mm/aa hora:mm:ss>

< tipo de movimentação> ::= "migrar" | "mover" | "replicar"

A chave de acesso será o nome do agente. Por cada registro de um agente, haverá um sub-registro por cada agente-filho, o qual terá os campos: Nome do agente-filho e o nome da agência destino, ambos do tipo *string*.

6.3.2. Serviço de Nomes

Este serviço [Naming97] é utilizado para registrar os objetos-CORBA que representam as agências. A partir dele, pode-se obter o endereço das mesmas e assim, acessar cada uma das interfaces públicas que elas possuem. Como o itinerário dos agentes vai estar baseado nos nomes das agências, este serviço devolverá uma referência ao objeto que representa a agência do itinerário.

6.3.3. Serviço de Propriedades

Este serviço [Property97] pode ser utilizado para padronizar o acesso aos arquivos meta-dados do agente e permitir, no futuro, a implementação de ferramentas baseadas neste serviço para a manipulação dos agentes. Portanto, sua utilização pelo STA e o SRA é uma questão opcional para o implementador.

6.3.4. Serviço de Eventos

Na realidade, o uso deste serviço [Event97] é uma questão do implementador, mas neste caso é necessário que seja usado para controlar os eventos gerados pelas invocações do cliente do STA e os eventos gerados pelas agências-destino, as quais informam sobre o resultado da transferência. Desta forma, está-se cumprindo com a característica de assincronismo na operação da agência. Evidentemente, poderá ser usado em outras etapas internas da agência.

Deficiências deste serviço: Na forma em que este serviço está atualmente especificado, o Canal de Eventos carece das seguintes facilidades:

1. Não possui filtros para enviar os eventos de forma seletiva aos consumidores. Ou seja, os consumidores não podem indicar ao canal que tipo de eventos esperam receber. Atualmente, o canal envia o evento (*push model*) a todos os consumidores, e estes devem decidir se é o tipo de evento que esperavam. Ou então o consumidor pega um evento do canal (*pull model*) sem saber se realmente é para ele mesmo, decidindo posteriormente sobre o seu uso.
2. Não permite implementar uma política de correlação de eventos, isto é, uma política que permita entregar um evento ou um conjunto de eventos ao consumidor, quando se cumpra certa relação lógica entre os eventos no canal.
3. Não especifica a forma de fazer o despacho dos eventos (escalonamento baseado em prioridades).

Mas cumpre com a função básica de comunicação assíncrona entre os consumidores e produtores, sem modificar ou estender a arquitetura CORBA. Atualmente, um serviço de Notificação [Notification98], que está orientado a complementar o serviço de Eventos, abordando as questões anteriormente mencionadas, foi aprovado na OMG para iniciar a etapa de testes e formalizar a especificação no final de Janeiro do 2000.

Também se especificou o Serviço de Messaging, atualmente na fase de testes de implementação [Messaging98], orientado a especificar um tipo de invocação totalmente assíncrona: espera-se formalizar a especificação no final de Maio do 2000. Esta especificação modificará os *stubs*, DII e ORB, para que possam gerenciar o envio das mensagens usando o esquema *forward*, além do que o cliente, no caso que precise desligar-se antes que chegue a resposta, deve deixar um representante para recebê-la.

6.4. Especificação das interfaces do STA e do SRA

Depois de ter descrito a arquitetura de uma agência para o suporte dos agentes móveis, vai-se definir um conjunto de interfaces que permitam a construção e interação dos diferentes módulos que compõem o Serviço de Transporte de Agentes.

Na figura 6.4, representam-se, por objetos computacionais, os módulos da arquitetura proposta, descrevendo-se as interações entre eles através das interfaces definidas. Na qual, apresentam-se 8 interfaces (cor cinza), as quais representam as interfaces internas do STA e onde sua definição depende do implementador. As 4 interfaces de cor preta, representam pontos do STA de acesso público e portanto, serão definidas utilizando a especificação IDL da OMG. Por questões de simplicidade, não se mostra a interação com os serviços de Registro de Agentes, de Nomes e de Eventos.

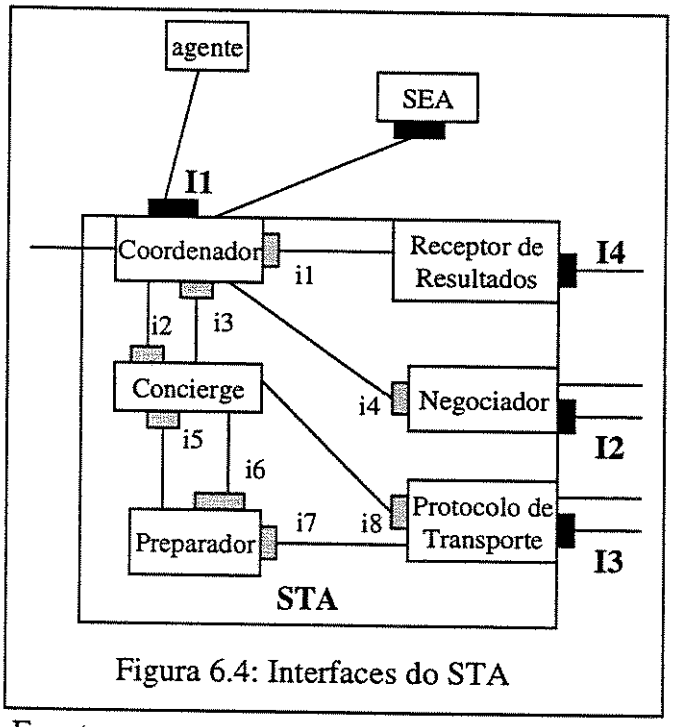

Figura 6.4: Interfaces do STA

A seguir, vai-se apresentar a especificação, em IDL, das interfaces para os serviços de transporte (STA) e registro dos agentes (SRA), os quais deverão comportar-se como foi descrito anteriormente.

Na figura 6.5, apresenta-se a especificação das interfaces do STA de acesso público, além da interface do SRA, sem considerar as exceções por motivos de simplicidade.

```
#include "CosEventComm.idl"

module MAFSystem {

/*****************************************************************/
/*                          Data Types                           */
/*****************************************************************/
enum Kind {AgentTransfer, RemoteResult, IIOPServer, Negotiate,
           MAFAgentRegistration}; // Representa o conjunto de nomes de interfaces
                                  da agência;
typedef sequence< octet > BinStream;
typedef BinStream AgentProfile;

typedef sequence< TypeProtocol> ProtProposal;
enum TypeProtocol {IIOP, FTP, SMTP};

typedef string PropertyName;
typedef string PropertyValue;

struct Property {
    PropertyName name;
    PropertyValue value;
};

typedef sequence<Property> PropertySeq;

struct Protocol {
  TypeProtocol protocol_name;  // Protocolo escolhido
  PropertySeq detail_tx_prot;  // Informação relativa ao protocolo escolhido
```

```
};

typedef string RefString;
enum ObjType {CORBA, RMI }; //Outros tipos de objetos serão acrescentados

/*****************************************************************/
/*              Definição das Interfaces                         */
/*****************************************************************/

interface MafFactory {
  Object create_object(in Kind interface_name);
};

interface AgentTransfer : CosEventComm:: PushSupplier {
void migrate(in string agent_name, in string destination);
void move(in string agent_name, in string destination);
void replicate(in string agent_name, in string destination, in string agent_name_jr);

};

interface Negotiate {
 boolean booking(in AgentProfile item_list, in ProtProposal item_prot,
                 in string source_agency_name, out Protocol protocol_selected);
};

interface IIOPServer : CosEventComm:: PushSupplier {
 boolean begin_tx();
 boolean tx_file(in BinStream data, in long length);
 boolean end_tx();
};

interface RemoteResult : CosEventComm:: PushSupplier {
```

```
  boolean result(in string agent_name, in string report);
 boolean replication_result(in string agent_name_jr, in string report_or_objref);

};

interface MAFAgentRegistration {
  boolean grant (in string ag_name);
  boolean change_situation(in string ag_name, in string new_ situation);
  boolean reservation(in string ag_name, in string work_dir,
                          in string source_ag_name, in KindMoving type_of_moving);
  boolean deregister(in string ag_name);
  boolean register_son(in string ag_name_jr, in string destination);
  boolean deregister_son(in string ag_name_jr);
 ObjType get_ref(in string ag_name, out RefString obj_ref );
};

}; // Fim do modulo MAFSystem
```

Figura 6.5: Especificação IDL das interfaces do STA e do SRA

De modo geral e para facilitar a manipulação destas interfaces, definiu-se uma interface-fabricante, denominada *MafFactory*, a qual permitirá criar cada uma das interfaces do STA e SRA, indicando o nome delas. Desta forma, uma agência será representada por um único objeto permanente, acessível através desta interface.

A seguir, descrevem-se cada uma das interfaces públicas.

6.4.1. Interfaces IDL para o STA

Para o módulo de Coordenação definiu-se a interface (I1) denominada ***AgentTransfer***, oferecendo aos clientes do STA (agentes e donos de agentes) as operações de movimentação: *migrate, move e replicate*, todas elas tendo o nome do agente e seu destino como parâmetros de entrada, mais o nome do agente-filho no caso do *replicate*.

Como o STA deve ser assíncrono, esta interface herda do Serviço de Eventos, a interface *PushSupplier* definida no módulo *CosEventComm*. Desta forma, uma vez que o agente ou o dono do agente faz a solicitação, pode continuar realizando outras atividades.

Para o módulo de Negociação definiu-se a interface (I2) denominada ***Negotiate***, oferecendo a operação *booking*, tendo como parâmetros de entrada: uma lista de itens, representado pelo tipo ***AgentProfile*** e outra lista do tipo ***ProtProposal***, que representa a lista de protocolos propostos: também deve-se indicar o nome da agência origem, para que o destino saiba como reenviar qualquer informação posteriormente. O tipo ***AgentProfile*** é uma seqüência de octetos, a qual é utilizada para enviar o arquivo "negotia.tor". O tipo *ProtProposal* define uma seqüência de números que representam os protocolos propostos na ordem de prioridades indicada. Definiu-se um parâmetro de saída para retornar uma estrutura de dados, denominada ***Protocol***, com os valores dos parâmetros necessários para usar o protocolo escolhido durante a negociação. Esta operação retorna um valor *booleano*, para indicar se a negociação teve sucesso.

No futuro, poderá ser considerada a possibilidade de que a agência destino, através desta operação, retorne como parâmetro de saída, uma chave pública para que a agência origem criptografe o pacote-receptáculo do agente antes de transmiti-lo.

Como o IIOP será o protocolo de transporte básico que toda agência vai ter, definiu-se a interface (I3) denominada ***IIOPServer***, oferecendo:

- A operação *begin*, sem parâmetros, para iniciar a sessão de transferência;
- A operação *tx_file* para transferir o pacote-receptáculo do agente, tendo como parâmetros de entrada uma *string* de otetos e o comprimento do dito *string* que está sendo transferido;
- E a operação *end_tx*, sem parâmetros, para indicar ao servidor que terminou a transferência do agente.

Finalmente, definiu-se para o módulo de Recepção de Resultado a interface (I4) denominada ***RemoteResult***, a qual oferece duas operações: *result* e *replication_result*, onde:

- A operação *result* é usada para enviar à origem o resultado de uma operação de movimentação de agente (migrar ou mover), tendo como parâmetros de entrada o nome do agente, especificando para qual agente pertence o resultado e uma *string* que explica os motivos no caso em que a movimentação do agente não venha a ter sucesso. Esta operação retorna um valor *booleano* de verdadeiro, caso contrário o valor de falso, fazendo relevante o segundo parâmetro de entrada.

- A operação *replication_result* é usada no caso das réplicas dos agentes, a qual tem dois parâmetros de entrada que representam: o nome do agente-filho e um parâmetro, cujo valor dependerá do resultado da operação de réplica, refletido no valor *booleano* que retorna esta operação. Se a réplica não tiver sucesso (retorna o valor *booleano* de falso), então o segundo parâmetro indicará os motivos. Se tiver sucesso, o conteúdo deste parâmetro será a referência a uma interface de gerenciamento do agente-filho.

Uma outra alternativa era definir uma única operação *result* para esta interface, deixando que a operação pegue o nome do agente para verificar se trata-se de um pai ou filho (detectando o separador "+"), determinando assim se corresponde a uma migração/movimentação ou a uma réplica de agente. Só que, se a réplica de um agente decide migrar ou mover-se, esta operação vai achar, pelo nome do agente, que se trata de uma réplica, levando a erro. Portanto, preferiu-se a primeira alternativa por ser mais clara na apresentação do serviço e evitando ambigüidades na sua execução.

Através da interface *RemoteResult* o ciclo do processo assíncrono de movimentação é fechado: ela herda a interface *PushSupplier* do módulo *CosEventComm* do Serviço de Eventos, de tal forma que, quando receber uma resposta, será deixada no canal de eventos.

6.4.2. Interface IDL para o SRA

Para este serviço definiu-se uma interface oferecendo um conjunto de operações necessárias para o funcionamento do STA. No futuro, outras operações poderão acrescentar-se, para preencher as necessidades dos outros serviços da agência. As seguintes são as operações definidas até agora:

- A operação *grant*, a qual permite consultar à agência se o agente tem reserva. Tem o nome do agente como único parâmetro entrada e devolve o valor de "verdadeiro", se a reserva para o agente existe.

- Para mudar a situação de um agente utiliza-se a operação *change_situation*, na qual deve-se indicar, como parâmetros de entrada, o nome do agente e a nova situação do mesmo.

- Para solicitar uma reserva para um agente, oferece-se a operação *reservation* indicando como parâmetros de entrada: o nome do agente, nome do diretório de trabalho assinalado pela agência, nome da agência origem e tipo de movimentação. Retorna um valor *booleano* para indicar o resultado da operação.

- Para manipular os agentes-filhos: define-se a operação *register_son* no registro do agente-pai, junto com o nome do destino.

- Para retirar o registro de um agente e seus componentes: no caso de o agente não continuar na agência, definiu-se a operação *deregister*, a qual precisa do nome do agente como parâmetro de entrada, retornando um valor *booleano* para indicar o resultado da operação.

- Para obter a referência para um agente, definiu-se a operação *get_ref*, a qual devolve uma referência como uma *string*. O cliente deverá saber como obter realmente a referência a partir deste dado. Por exemplo, no caso em que o agente seja um objeto Corba, a referência terá o formato IOR, e o cliente deverá aplicar a operação *stringified* para obter a referência ao agente. Por este motivo, utiliza-se o parâmetro de retorno que identifica o tipo de objeto (por exemplo: CORBA).

6.5. Resumo

Neste capítulo descreveu-se a operação do STA através de dois cenários: despacho e recepção de um agente.

Também, definiu-se e explicou-se o formato do receptáculo para conter os componentes de um agente, e que será extraído e instalado pela agência destino. Aqui destaca-se a utilização de XML para descrever os requerimentos para a instalação e execução dos agentes e das propriedades do mesmo. Estas descrições serão salvas em arquivos que serão transferidos para a negociação prévia do agente.

Na especificação, estabelece-se que a negociação será feita através de uma interface, onde os parâmetros conterão arquivos com as descrições em XML. Entretanto, a interface de transporte especifica como transferir um agente utilizando as invocações CORBA, ou seja, especifica como transferir o receptáculo que contém um agente, usando IIOP como protocolo de transporte. Esta especificação deixa aberta a possibilidade de utilizar outros protocolos de transporte como TCP, RMI e outros.

Analisa-se a utilização dos serviços CORBA de: Nomes e Eventos, achando-se eles adequados para gerenciar os nomes das agências e para que o STA seja implementado como um serviço assíncrono.

Justifica-se a criação do Serviço de Registro de Agentes, para que seja utilizado pelo STA durante o gerenciamento do agente. Este serviço será especificado posteriormente.

Baseado nas considerações anteriores, finalmente, especificam-se as interfaces IDL de um STA.

No capítulo seguinte, serão abordadas as questões associadas à implementação deste serviço.

7. Aspectos de implementação de um STA

Após ter-se analisado o STA nos capítulos anteriores, desenvolveu-se um projeto com os seguintes objetivos:

- Determinar as partes que deveriam compor este serviço, considerando suas interfaces IDL.
- Implementar e testar as partes mais importantes do serviço, que tenham relação direta no desempenho.
- Mostrar uma visão prática sobre as dificuldades técnicas que podem encontrar-se na implementação deste serviço.

O capítulo foi estruturado em três partes:

- Na primeira, abordar-se-á uma análise do projeto, refinando o diagrama geral apresentado no capítulo anterior, baseado nos *use cases* e diagramas de colaboração, para determinar o diagrama de classes e suas relações para o STA, sem considerar as interfaces nem o serviço de eventos.
- Na segunda parte, acrescentar-se-ão ao modelo de classes: as interfaces IDL do STA e do serviço de eventos, tendo assim um modelo completo do projeto de um STA.
- Na terceira parte, descrever-se-á uma implementação feita para testar algumas partes referentes ao transporte de um agente.

O projeto foi desenvolvido utilizando como ferramenta UML e o pacote denominado *Rose98* da empresa "Rational Software Co." [Rational98]. A implementação foi feita utilizando Java versão 1.02, como a linguagem de programação e Visibroker for Java, versão 3.2, da empresa Inprise [Inprise99], como plataforma ORB e incluindo o Serviço de Eventos.

7.1. Projeto de um STA

O projeto iniciou-se elaborando vários "casos de uso" do sistema ou agência. Em primeiro lugar, determinaram-se dois tipos de atores: o agente ou seu dono e a agência propriamente dita. O agente utilizará a agência para migrar, mover-se ou replicar-se. A agência vai contatar o sistema ou agência remota em dois casos: quando solicitar permissão para enviar o agente e quando tiver que despachá-lo. No projeto a agência foi denominada MAFSystem (Sistema de Facilidades para Agentes Móveis)

No primeiro caso o sistema será usado para migrar, mover e replicar um agente, no segundo, para processamento da permissão e para receber o agente. Logo, cinco casos de uso terão que ser definidos: Migrar, Mover, Replicar, Permissão e Receber(Figura 7.1). Mas, nos três primeiros casos, encontrou-se que eles têm comportamento comum quando negocia e despacha o agente; gerando-se os novos casos de uso: Negociar e Despachar.

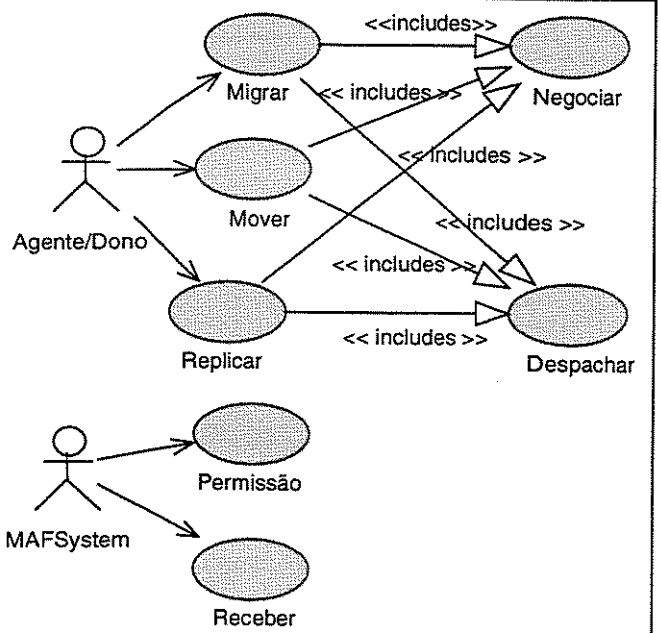

Figura 7.1: Casos de uso do STA

Com ajuda dos diagramas de colaboração para os diferentes casos de uso, foi possível determinar os objetos que neles participam e as relações entre eles quando os casos de uso são analisados. Como exemplos, as Figuras 7.2 e 7.3, apresentam os diagramas de colaboração para os casos de uso: Migrar e Despachar. Estes diagramas, junto com outros que foram desenvolvidos para os restantes casos de uso, permitiram construir os diversos diagramas de classes e suas relações que descrevem um STA.

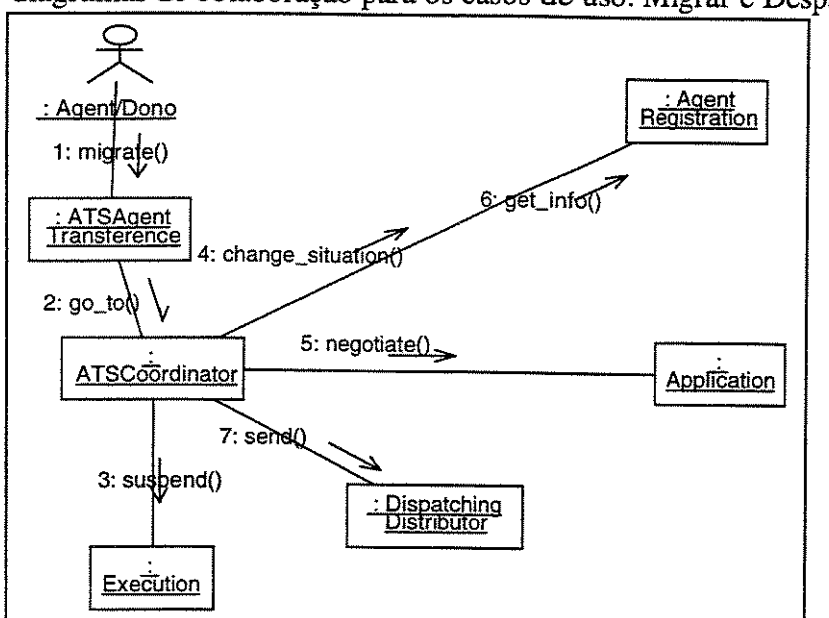

Figura 7.2: Diagrama de colaboração para o caso de uso Migrar

Para facilitar a apresentação, o sistema do STA foi dividido em três pacotes, denominados: *MobilitySystem, TransportSyste e, NegotiationSystem.* Além dos anteriores, foram definidos os pacotes *RegistrationSystem* e *System*, o primeiro para descrever o Serviço de Registro do Agente (SRA) e o segundo para indicar outros sistemas que o STA utiliza e que não são relevantes (por ex.: o sistema operacional (*OperatingSystem*) e o Serviço de Execução de Agentes (*Execution*)).

No pacote *MobilitySystem*, apresentado na Figura 7.4, representa-se o sistema sob o ponto

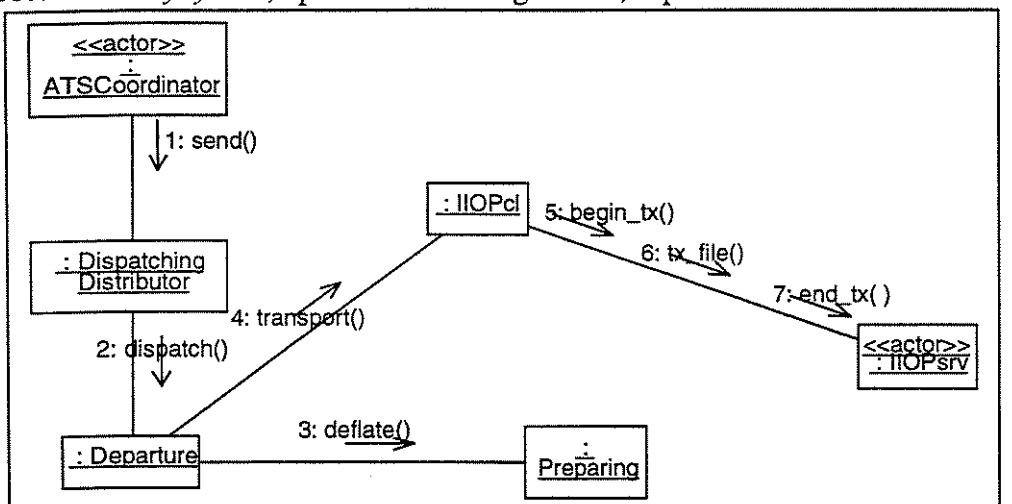

Figura 7.3: Diagrama de colaboração para o caso de uso Despachar

de *vista superior*, destacando-se a interação do agente com o coordenador, através da classe *ATSAgentTransference*, a relação do coordenador com as outras classes para coordenar as diversas tarefas do STA, e incluindo-se as classes *ATSRemoteResult* e *Finishing*, encarregadas de concluir a transferência assíncrona do agente.

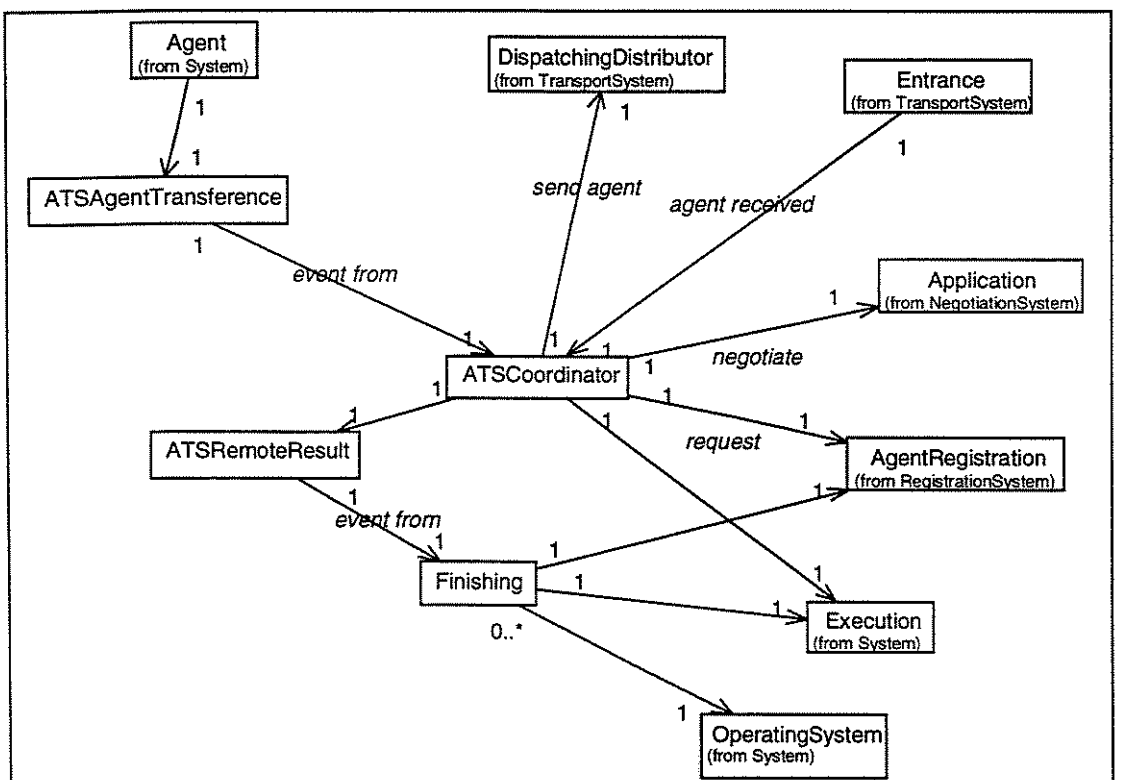

Figura 7.4: Diagrama de classes para a coordenação da mobilidade (pacote Mobility System)

No pacote *TransportSystem*, apresentado na Figura 7.5, modelam-se as diferentes classes e suas relações, que participam no mecanismo do transporte. Os pares de classes

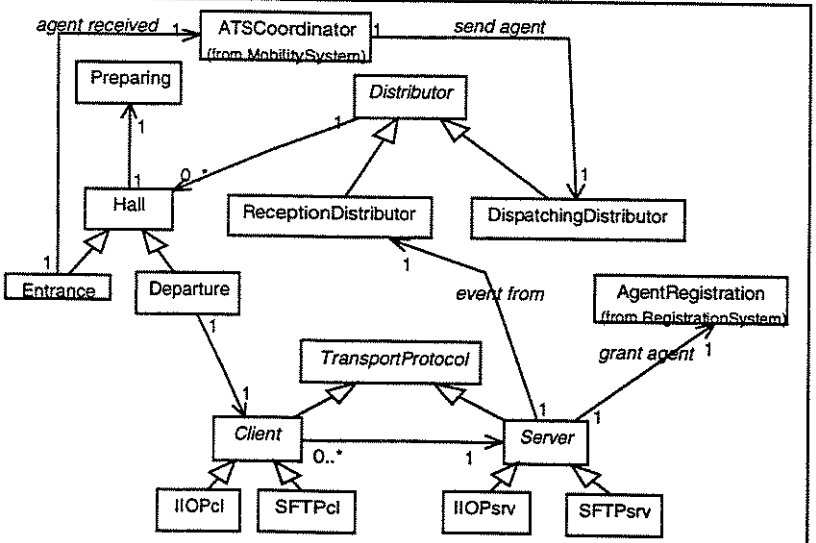

Figura 7.5: Diagrama de classes para o sistema de transporte (pacote Transport System)

DispatchingDistributor – Departure e *ReceptionDistributor – Entrance*, respetivamente representam os objetos responsáveis pelo despacho e recepção do agente. As classes: *DispatchingDistributor* e *ReceptionDistributor*, pelas suas características de distribuição, foram relacionadas pela super-classe *Distributor*. Entretanto, as classes *Departure* e *Entrance*, foram relacionadas pela superclasse *Hall*, porque têm características comuns para os agentes que saem e entram da agência. Os objetos da classe *Preparing* estão preocupados em codificar/decodificar e comprimir/descomprimir os componentes dos agentes, etapas prévias no despacho e recepção de um agente.

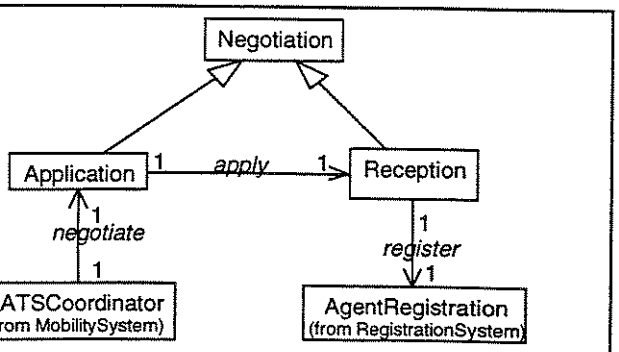

Figura 7.6: Diagrama de classes para a negociação (pacote NegotiationSystem)

Para o transporte propriamente dito de um agente, tem-se as classes *Client – Server*, que representam o conjunto de protocolos alternativos para o transporte, relacionadas pela superclasse *TransportProtocol*, pelas características comuns que ambas possuem. A partir delas, podem-se definir as classes que representam os diferentes protocolos de transporte (IIOP, FTP, SMTP, etc.) que uma agência suporta.

No pacote *NegotiationSystem*, apresentado na Figura 7.6, descreve-se o modelo representando as classes envolvidas na etapa de negociação para solicitar permissão e enviar um agente. Aqui as classes mais relevantes são *Application* e *Reception*,

representando os objetos que participam na negociação entre as agências fonte e destino. Basicamente, a informação requisitada para executar o agente é enviada por um objeto da classe *Application* para um objeto da classe *Reception*, no destino. Se o agente é aceito, uma reserva será mantida até sua chegada. Pelas suas características de negociação, elas foram relacionadas pela superclasse *Negotiation*.

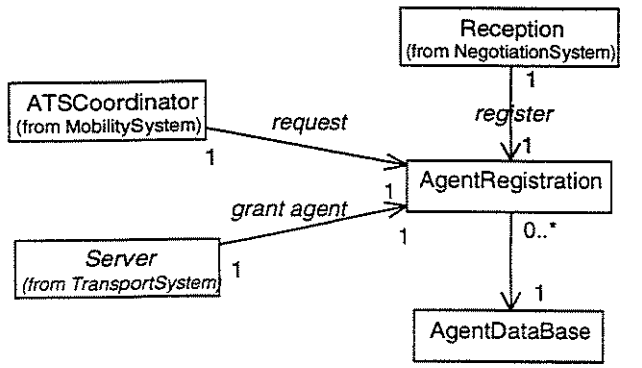

Figura 7.7: Diagrama de classes para as atividades de registro de um agente (pacote RegistrationSystem)

No pacote *RegistrationSystem*, apresentado na Figura 7.7, descreve-se a relação da classe *AgentRegistration*, representando o Serviço de Registro de Agentes, com a classe *AgentDataBase*, para representar sua relação com uma base de dados genérica, além de se mostrar sua relação com as outras classes do STA que a usam.

7.2. Aspectos de implementação

Nesta etapa, foram acrescentados aos modelos de diagrama de classes do STA, os aspectos

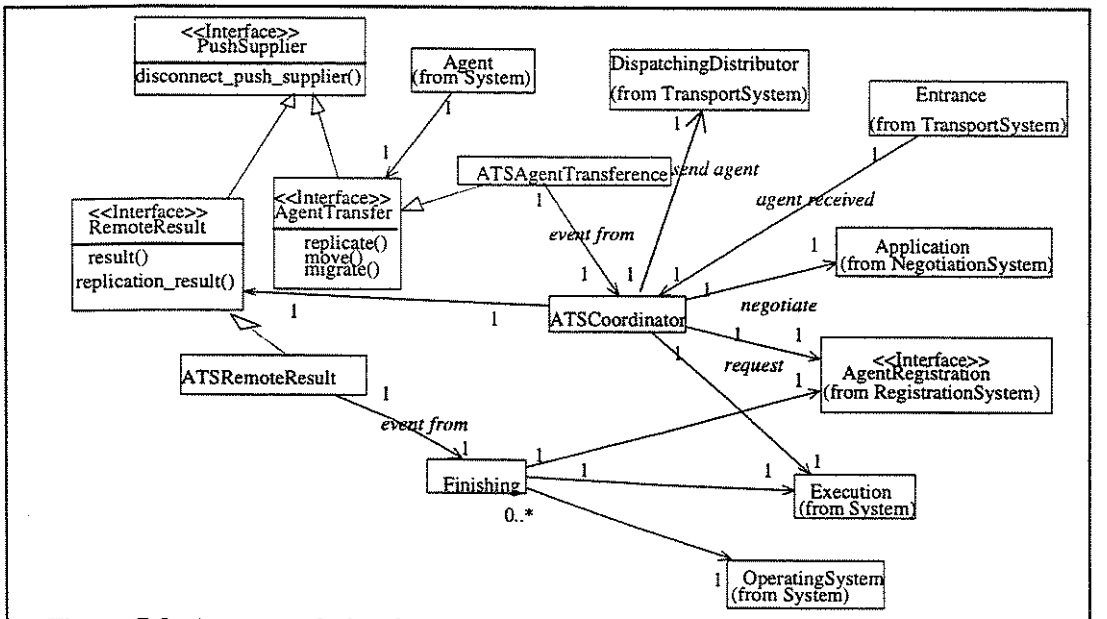

Figura 7.8: Aspectos de implementação para a coordenação na agência

de implementação, como: a relação do STA com o Serviço de Eventos e com as interfaces IDL definidas no capítulo anterior. Os diagramas de classes que expressam estes aspectos de implementação são apresentados nas Figuras, 7.8, 7.9, 7.10 e 7.11.

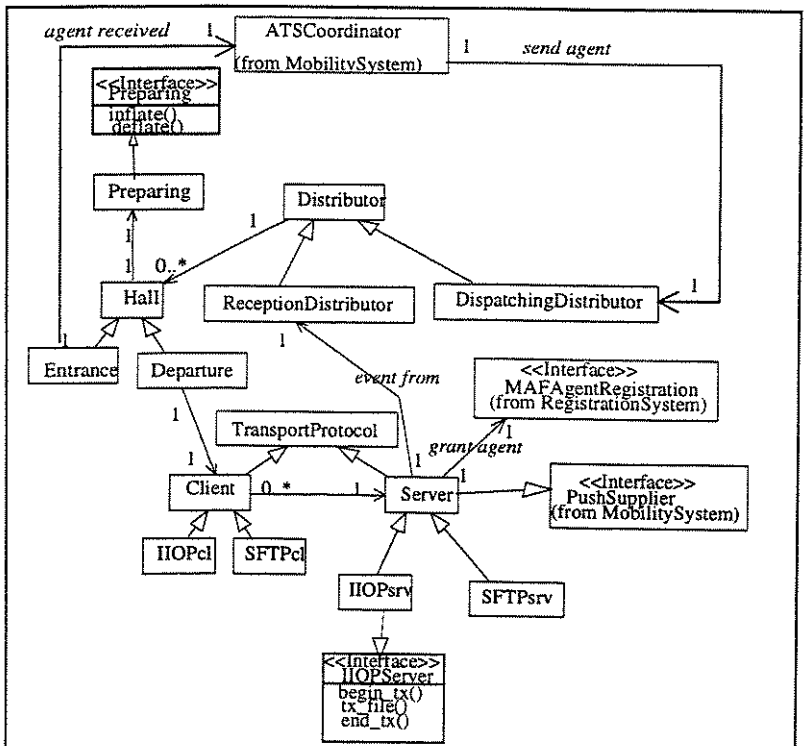

Figura 7.9: Aspectos de implementação para o sistema de transporte

No caso do Serviço de Eventos, este ficou representado pelas interfaces: *PushConsumer* e *PushSupplier*. Através destas interfaces, o STA poderá colocar num canal de eventos, através da interface *PushSupplier* os eventos gerados durante o processo de transferência do agente ou processar os eventos, recebidos pelo canal de eventos, através da interface *PushConsumer*.

Nesta etapa, foram acrescentadas ao modelo só as interfaces IDL do STA e expressando a relação que elas têm com as interfaces do Serviço de Evento. Por exemplo, as interfaces IDL do STA: *RemoteResult*, *AgentTransfer* e *IIOPServer*, herdam a interface *PushSupplier*, porque durante o processo de transferência de um agente, geram eventos, fazendo que o STA opere de forma assíncrona. A interface *PushConsumer* não aparece porque é questão do implementador definir quais objetos da agência consumirão os eventos, dependendo muito da distribuição dos mesmos, como se mostrará no exemplo de implementação.

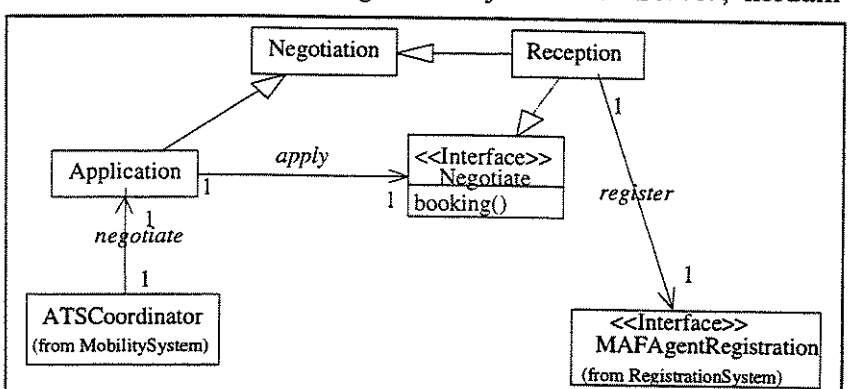

Figura 7.10: Aspectos de implementação para a negociação do agente

Outras interfaces IDL poderão ser acrescentadas dependendo da distribuição que o implementador queira dar à construção da agência. Isto será salientado quando se apresentar o exemplo de implementação.

Também não foram descritas, nestes diagramas, as classes que representam os objetos

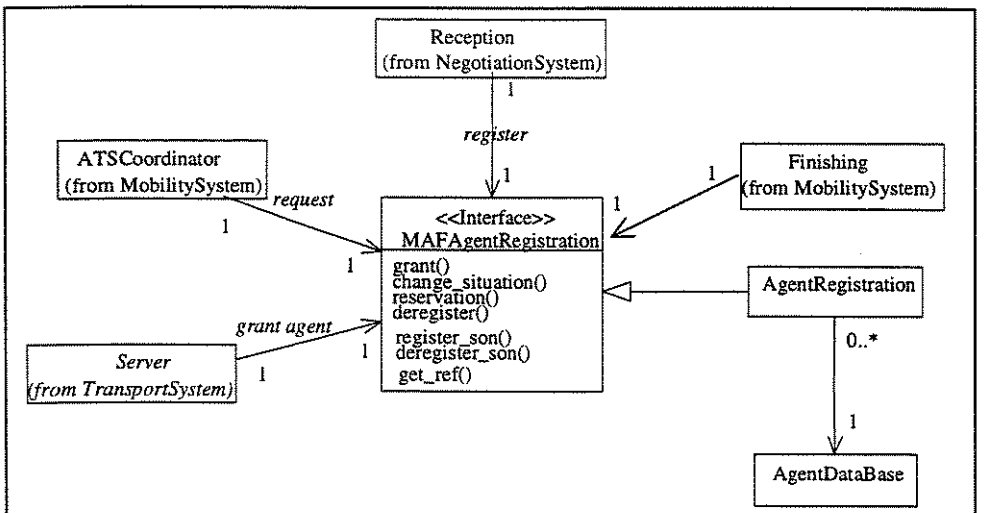

Figura 7.11: Aspectos de implementação para o sistema de registro dos agentes

fabricantes de objetos CORBA, por ser decisão do implementador o tratamento da questão da criação dos objetos, mesmo que ele decida implementar o sistema sem usar CORBA.

7.3. Um exemplo de implementação

Neste exemplo vai-se testar parte do sistema de transporte (*TransportingSystem*) para realizar a transferência de um agente utilizando IIOP ou invocações CORBA. Para o exemplo assume-se que:

- A negociação foi feita com sucesso;
- Não se considera a codificação e decodificação do pacote do agente;
- Considera-se a compressão e descompressão do diretório de trabalho do agente; utilizando o formato *Zip*, o qual será transportado como um pacote.

Neste cenário, vai existir um lado "Local" e outro "Remoto". O lado "Local" vai ter quatro unidades de implementação ("A", "B", "C" e "D") e o lado "Remoto" terá quatro unidades de implementação ("E", "F", "G" e "H"), as quais podem compartilhar um computador ou serem distribuídas em mais de um. As unidades são implementações dos módulos de programas, que abrangem um ou mais objetos de implementação de objetos CORBA e de objetos próprios da linguagem Java. Pelo menos um deles deverá ser um objeto CORBA permanente, que ficará ativo à escuta de requerimentos. Em cada lado haverá unidades que implementam o Serviço de Eventos (unidades "B" e "F") sobre as quais, só é necessário dizer-se que serão formadas pelos objetos CORBA permanente "Local" e "Remote", criados quando o serviço é ativado.

Na implementação, determinou-se que os objetos das classes *ReceptionDistributor* e *DispatchingDistributor*, serão as consumidoras dos eventos relacionados com a solicitação de receber ou despachar um agente. Para isso estas classes herdarão a interface *PushConsumer*, modificando o diagrama de classes para o pacote *Transporting System* da forma como se apresenta na Figura 7.12.

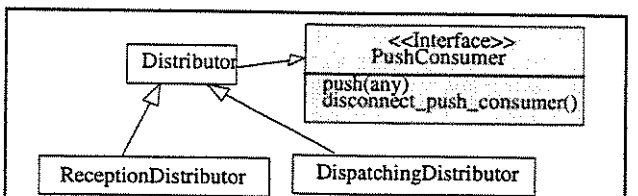

Figura 7.12: Definição arbitraria dos consumidores no sistema de transporte

Este cenário é apresentado na figura 7.13, onde todo objeto que seja mencionado neste

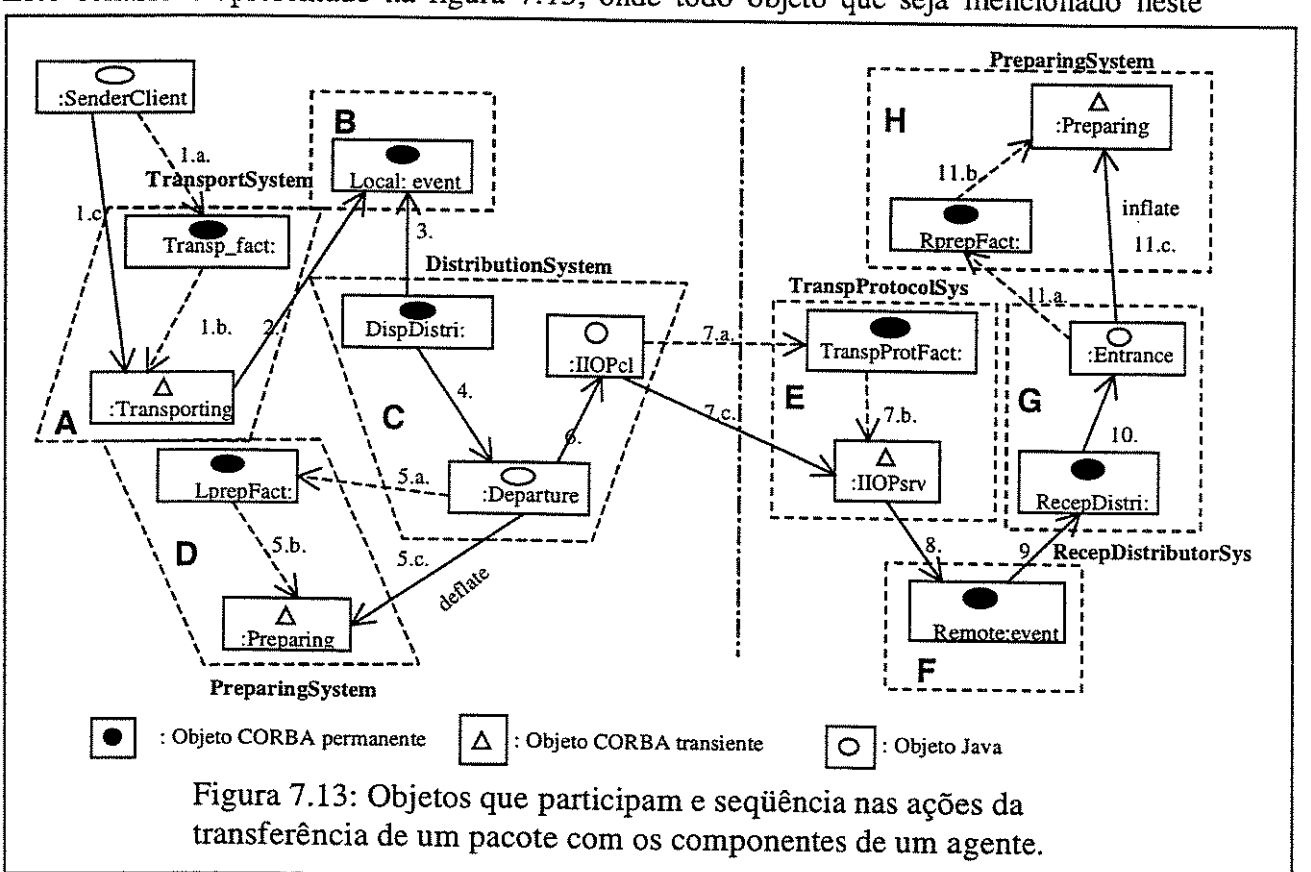

Figura 7.13: Objetos que participam e seqüência nas ações da transferência de um pacote com os componentes de um agente.

exemplo será encontrado na figura. No caso dos objetos permanentes, eles serão indicados por seu nome.

A listagem, com a especificação IDL das interfaces definidas para o exemplo, é apresentada na figura 7.14.

7.3.1. Tratamento dos eventos

No cenário acontecerão dois tipos de eventos, que serão identificados pelas estruturas de dados: *ParamList*, definida no módulo *DistributionSystem*, e *EntranceList*, definida no módulo *TranspProtocolSys*.

O evento ligado à *ParamList*, acontecerá no lado "Local" quando uma solicitação para transferir um agente seja feita pelo cliente, representado pelo objeto Java do tipo *SenderClient*. O dado será colocado no canal de eventos pelo objeto do tipo *Transporting*, para que seja processado pelo objeto "DispiDistri".

O outro evento acontecerá no lado "Remoto", quando o pacote contendo os componentes de um agente seja recebido. Neste caso, o objeto do tipo *IIOPsrv*, colocará, no canal de eventos, um dado com a estrutura definida por *EntranceList*, para que seja processada pelo objeto "RecepDistri".

Estas estruturas de dados não estão padronizadas e dependem do implementador da agência.

7.3.2. Descrição das diversas partes que compõem o cenário

<u>No lado "Local"</u>:

Para ter acesso ao serviço implementou-se, na unidade "A" (módulo *TransportSystem*), uma interface IDL denominada *Transporting*, implementada por um objeto CORBA transiente, que não é parte do STA e que foi criado só com objetivo de acessar e testar o transporte de agentes. Esta interface herda a interface *PushSupplier* para que a solicitação para transferir o agente fique no canal de eventos. O objeto CORBA que implementa esta interface e que funciona como objeto transiente, será criado pelo objeto – fabricante cuja interface IDL está definida por *TranspSysFactory* (objeto "Transp_ fact"), que funciona como objeto CORBA permanente.

A unidade "C" (módulo *DistributionSystem*) abrange três objetos:

- Um objeto CORBA permanente ("DispDistri"), cuja interface IDL está definida por *DispatchingDistributor*. Na realidade, esta interface representa a classe *DispatchingDistributor* do pacote *TransportingSystem*, e que neste cenário foi

considerada como objeto CORBA para que possa ser distribuída no sistema. Portanto, esta foi uma decisão específica do exemplo e que não é mandatória.

- Um objeto Java da classe *Departure*.
- Um objeto Java da classe *IIOPcl*

Nesta unidade a eleição dos objetos como objetos CORBA ou objetos Java, foi totalmente arbitrária porque na especificação do STA não estão definidas como interfaces públicas IDL.

A unidade "D" (módulo *PreparingSystem*) abrange dois objetos:

- Um objeto CORBA permanente ("LprepFact"), que implementa um objeto – fabricante cuja interface IDL está definida por *PrepSysFactory*, para criar objetos do tipo *Preparing*.
- Um objeto CORBA transiente, cuja interface IDL é implementada pela classe *Preparing*.

Estes objetos não são parte das interfaces públicas do STA e são próprias desta implementação.

No lado "Remoto":

A unidade "E" (módulo *TranspProtocolSys*) implementa o lado servidor do protocolo de transporte IIOP utilizado para a transferência do agente. Esta unidade tem dois objetos CORBA:

- Um objeto CORBA permanente ("TranspProtFact"), que fica à escuta e encarregado de criar os objetos transientes do tipo *IIOPsrv*.
- Um objeto CORBA transiente do tipo *IIOPsrv*, encarregado de receber o agente e de avisar ao receptor, através do canal de eventos, que um agente chegou.

Neste caso, o objeto do tipo *IIOPsrv* corresponde a uma interface IDL pública do STA e a interface IDL do fabricante foi acrescentada à implementação.

A unidade "G" (módulo *RecepDistributorSys*) possui dois objetos:

- Um objeto CORBA permanente ("RecepDistri") cuja interface está definida por *ReceptionDistributor*, implementada pela classe do mesmo nome do pacote *TransportingSystem*, e ligado ao canal de eventos como consumidor, para receber o evento indicando que um pacote foi recebido e está pronto para ser desempacotado.
- Um objeto Java da classe *Entrance* do pacote *TransportingSystem*.

Estes objetos não são parte das interfaces públicas IDL do STA e portanto, sua implementação é própria deste exemplo.

A unidade "H" (módulo *PreparingSystem*) contém dois objetos CORBA, de forma similar à unidade "D" do lado "Local":

- Um objeto CORBA permanente ("RprepFact"), definida pela interface IDL *PrepSysFactory*.
- Um objeto CORBA transiente do tipo *Preparing*, que é criado por "RprepFact" para atender as solicitações do objeto do tipo *Entrance*.

Tendo descrito cada um dos objetos que participam neste cenário, vai-se descrever agora, a seqüência de execução para a transferência do pacote de um agente, de acordo com o representado na figura 7-13.

7.3.3. Seqüência de execução

1.a. Um processo cliente invoca o objeto fabricante, denominado "Transp_fact", para que crie um objeto do tipo *Transporting*.

1.b. Um objeto do tipo *Transporting* é criado, e sua referência é repassada ao cliente.

1.c. O cliente invoca a operação do *Transporting*, indicando o nome do agente, seu novo destino, seu diretório de trabalho e o protocolo de transporte selecionado.

2. O *Transporting* coloca os dados dentro de uma estrutura para que seja inserido no canal de eventos.

3. O objeto "DispDistri" pega o evento e pela estrutura do dado descobre que é para ele.

4. O objeto "DispDistri" cria um *thread* do tipo *Departure*, para que se encarregue de coordenar o transporte do agente.

5.a. O *Departure* se liga ao objeto "LprepFact", para solicitar-lhe que crie um objeto do tipo *Preparing*.

5.b. Um objeto do tipo *Preparing* é criado e sua referência é repassada ao *Departure*.

5.c. O *Departure* invoca a operação *deflate* para solicitar que o diretório de trabalho do agente seja comprimido.

6. O *Departure* solicita ao objeto do tipo *IIOPcl* que transporte o pacote comprimido do agente.

7.a. O objeto do tipo *IIOPcl* deve estabelecer uma ligação com o servidor do protocolo IIOP. Para isso, liga-se com o fabricante, denominado "TranspProtFact".

7.b. O fabricante cria um objeto do tipo *IIOPsrv*, devolvendo sua referência ao objeto cliente.

7.c. O objeto cliente procede a transmitir o pacote que contém os componentes do agente.

8. Após ter recebido o pacote, o objeto CORBA do tipo *IIOPsrv* coloca a informação sobre o agente que chegou, dentro do canal de eventos, para que seja processado pelo receptor.

9. O objeto CORBA denominado "RecepDistri" pega o evento (uma estrutura de dados com a informação sobre um agente)

10. "RecepDistri" procede a repassar a informação a um objeto Java do tipo *Entrance*.

11.a. O objeto do tipo Entrance liga-se a "RprepFact", um fabricante de objetos do tipo *Preparing*.

11.b. "RprepFact" cria um objeto CORBA do tipo *Preparing* e devolve a referência ao cliente.

11.c. O objeto do tipo *Entrance* solicita ao objeto transiente do tipo *Preparing* para que proceda a descomprimir (*inflate*) o pacote com os componentes do agente.

7.3.4. Ambiente de teste

Este cenário pode testar-se executando os diferentes programas que formam este exemplo. Para isso, deve-se considerar um ambiente de teste onde tem-se duas redes locais: uma considerada como "Local" e outra como "Remota", onde serão instalados os arquivos que contêm os programas descritos no exemplo.

Assume-se que em cada rede está instalado o ORB do Visibroker e que foi executado o comando "osagent" na linha de comandos de cada ambiente, o que permite ativar os ORBs para que fiquem à escuta das invocações e prontos para resolver os nomes dos objetos CORBA envolvidos.

A seqüência de comandos para testar este exemplo é a seguinte:

No lado "Local":

```
1. >vbj com.visigenic.vbroker.services.CosEvent.Channel Local
```

Isto ativará o canal de eventos com o nome de "Local".

```
2. >vbj TransportSystem.ServerTransporting Transp_fact
```

Será ativado o objeto CORBA fabricante, denominado "Transp_fact", que será usado pelos clientes para criar objetos do tipo *Transporting* e usá-lo para enviar um agente. O programa cliente "SenderClient.java" está codificado para se ligar a um objeto CORBA com esse nome. No caso de modificar o nome, haveria que modificar o lugar do programa cliente onde se referencia este objeto.

```
3. >vbj DistributionSystem.ServerDispatching
```

Cria-se um objeto CORBA permanente, denominado implicitamente no programa como "DispiDistri" e que fica ligado automaticamente ao canal de eventos, denominado "Local", como consumidor de eventos, à escuta dos requerimentos para transferir agentes.

```
4. >vbj PreparingSystem.ServerPreparing LprepFact
```

Cria-se um objeto fabricante, denominado "LprepFact", utilizado pelos *threads* do tipo *Departure*.

No lado "Remoto":

```
5. >vbj com.visigenic.vbroker.services.CosEvent.Channel
Remote
```

Ativa-se o canal de evento com o nome de "Remote".

```
6. >vbj TranspProtocolSys.ServerTranspProtocol TranspProtFact
```

Cria-se um objeto CORBA fabricante, denominado "TranspProtFact", que servirá para instanciar objetos servidores do protocolo IIOP, utilizado para transportar os componentes do agente até o destino.

```
7. >vbj PreparingSystem.ServerPreparing RprepFact
```

Idem ao passo 4, mas no lado "Remoto".

```
8. >vbj RecepDistributorSys.ServerReception RecepDistri
```

Ativa-se o objeto "RecepDistri", à escuta no canal "Remote" para a recepção dos componentes de um agente.

No lado "Local":

```
9. >vbj TransportSystem.SenderClient
```

Ativa-se a seqüência completa de eventos. Como conseqüência, um arquivo "Ag001.zip" aparecerá no subdiretório "\proyecto\Agents\Ag001\", o qual contém os arquivos comprimidos do subdiretório "Ag001". Este arquivo será transmitido para o lado "Remoto", onde será armazenado no subdiretório "\proyecto\Agents\D001\" com o nome de "D001.zip", o qual será descomprimido para armazenamento de seu conteúdo no subdiretório.

No exemplo, estes nomes devem manter-se, para não perder a consistência entre os programas.

```
// PushConsumer.idl

#pragma prefix "omg.org"

        exception Disconnected{};

        interface PushConsumer {
                void push(in any data) raises(Disconnected);
                void disconnect_push_consumer();
        };
```

```
// PushSupplier.idl

#pragma prefix "omg.org"

        interface PushSupplier {
                void disconnect_push_supplier();
        };
```

```
//DispatchingDistributor.idl

#include <PushConsumer.idl>

module DistributionSystem {

        struct ParamList {
                string dest;
                string a_name;
                string w_dir;
                string protoc;
        };

        interface DispatchingDistributor: PushConsumer {

        };
};
```

```
// Preparing.idl


module PreparingSystem {
```

```
        interface Preparing;

        interface PrepSysFactory {
        Preparing create_preparing();
        };

        interface Preparing {

        // Operations
        boolean deflate(in string work_dir);

        boolean inflate(in string pathCte, in string file_name);

        };

};
```

```
//ReceptionDistributor.idl

#include <TranspProtocol.idl>
#include <PushConsumer.idl>

module RecepDistributorSys {

        typedef TranspProtocolSys::EntranceList ReceptionList;

        interface ReceptionDistributor: PushConsumer {

        };
};
```

```
// Transporting.idl


#include <PushSupplier.idl>
module TransportSystem {

        interface Transporting;

        struct ParamList {
                string dest;
                string a_name;
                string w_dir;
                string protoc;
        };
```

```
        interface TranspSysFactory {
        Transporting create_transporting();
        };


        interface Transporting: PushSupplier {

        // Operations

        boolean send(in string destination, in string agent_name, in string
        work_dir, in string protocol);

        };

};
```

```
// TranspProtocol.idl

#include <PushSupplier.idl>

module TranspProtocolSys {
      typedef sequence< octet > BinStream;

        struct EntranceList {
                string a_name;
                string path_base;
                string w_dir;
        };

      interface IIOPsrv: PushSupplier  {
            boolean begin_tx();
            boolean tx_file(in BinStream data, in long length);
            boolean end_tx();
      };

      interface TranspProtocolFactory {
            IIOPsrv create_IIOP_server();
          };


};
```

Figura 7.14: Especificação IDL para o exemplo de implementação de um transporte de agentes.

7.4. Resumo

Neste capítulo, fez-se um exercício de implementação de uma parte do STA que foi especificado no capítulo anterior.

É importante salientar que durante a implementação tomaram-se decisões sobre a especificação de interfaces que permitem distribuir uma agência, o que poderá ser feito de forma arbitrária pelo implementador, baseado em seus próprios requerimentos e nos recursos de que disponha.

O projeto orientou-se à implementação do protocolo de transporte para a transferência de um agente, via IIOP, o qual foi testado sob um cenário previamente discutido. Destaca-se, nesta implementação, a utilização do Serviço de Eventos e a API que permite compactar o conteúdo da área de trabalho de um agente, a qual foi desenvolvida durante este trabalho, baseada nas API básicas sobre zip que o JDK oferece.

No próximo capítulo, será avaliado o desempenho do protocolo de transporte IIOP implementado, para tentar determinar as condições sob as quais uma aplicação baseada em agentes móveis possa ter um melhor desempenho que baseada no modelo do cliente/servidor.

Parte III: Discussão de resultados e comentários finais

8. Comparação dos modelos do Agente Móvel e do Cliente/Servidor

Até agora não existe um consenso na literatura sobre as vantagens do modelo do agente móvel sobre o modelo do cliente/servidor. Alguns autores, que afirmam a vantagem de um modelo sobre o outro [Chess97] [Vitek97] [Papaioannou98], não conseguem explicitar sob quais condições ela existirá.

Por exemplo, fala-se que o modelo do agente móvel facilitaria o desenvolvimento de aplicações em rede, pensando-se que, no futuro, esse será o grande cenário da programação das aplicações cada vez mais complexas. Mas tudo indica que as aplicações do modelo do agente móvel deverão esperar que o modelo de objetos distribuídos na rede se consolide, aproveitando-se desta plataforma para um desenvolvimento mais rápido.

Por outro lado indica-se, como vantagem evidente, o fato do modelo de agentes móveis permitir desenvolver aplicações que gerem menos tráfego [Villinger96] que o modelo C/S, em particular nos casos em a interação do cliente com diversos servidores, deva ser feita de forma seqüencial, como pode acontecer em uma aplicação de comércio eletrônico e de gerenciamento de sistemas workflow [Cai96]. Mas esta situação só seria válida num cenário de redes de alto tráfego onde os tempos de espera, pela transmissão dos dados, podem ter valores altos.

Um outro caso diz respeito a redes de alto desempenho, em que diversos autores [Strasser97] [Baumann97] usam o modelo do agente móvel para a execução com paralelismo de aplicações residentes em diversos servidores. Esta situação não é objeto do estudo da tese.

De qualquer forma, nos dois cenários anteriores, as vantagens seriam reais só sob certas circunstâncias, que terão que ser analisadas de forma explícita. É por este motivo que o presente capítulo abordará uma comparação dos modelos do agente móvel e do cliente/servidor, considerando um cenário de uma rede global de alto tráfego como é a Internet. Em particular procura-se determinar as condições sob as quais o modelo do agente móvel gerará menos tráfego [Baldi98], melhorando assim o desempenho da aplicação. O protótipo desenvolvido no trabalho e discutido nos capítulos anteriores será usado para essa análise, demonstrando-se simultaneamente a sua validade.

O capítulo foi estruturado em duas partes. A primeira aborda os modelos teóricos e a segunda apresenta os resultados experimentais obtidos a partir das medições dos tempos de transferência de dados nas redes Internet e intranet.

8.1. Comparação teórica dos modelos

Antes de iniciar a análise se tomaram em conta as seguintes considerações:

(a) No modelo do C/S o tempo que demora uma aplicação, para realizar seu trabalho, é independente do tamanho da mesma porque o(s) cliente(s) e o(s) servidor(es) são entidades estáticas. Entretanto no modelo do agente móvel, a demora depende diretamente do tamanho do agente, devido à transferência do agente até o destino, antes de ser executado.

(b) Nos artigos que fazem algum tipo de análise do agente móvel, todos eles estão baseados no tamanho do agente e da mala com os dados que irá coletando [Strasser97]. Contudo este parâmetro não foi incluído na análise do trabalho, porque os modelos são de natureza muito diferente.

Isto se explica melhor, pensando que a rede gera a janela de tempo mais importante a considerar, porque é nela que acontecem os eventos que influem diretamente no desempenho da aplicação, ou seja a rede global é o fator mais importante no baixo desempenho das aplicações em rede. No caso do modelo C/S esse fato traduz-se em tempos maiores das invocações, podendo-se desprezar o tempo de execução, tanto no lado cliente como no lado do servidor. No caso do agente móvel, é o tempo que demora o agente para ser transferido até seu destino, que deve ser considerado, podendo-se desprezar os tempos de execução das tarefas do agente.

A análise deste capítulo considera assim o tempo como o parâmetro de comparação dos dois modelos. Assim, no caso do agente, o seu tamanho será fator preponderante, enquanto que no caso do C/S, o tamanho das estruturas de dados transferidas terá maior influência. Além disso, a análise será baseada no tempo que cada estratégia demora em realizar seu trabalho sob condições iguais de tráfego na rede.

Nesta análise foram avaliadas três possibilidades de uma aplicação que consulta, de forma seqüencial, num conjunto de servidores:

- Modelo do cliente/servidor (figura 8.1);
- Modelo do agente móvel sem negociação (figura 8.2);
- Modelo do agente móvel com negociação (figura 8.3).

8.1.1. Modelo Cliente/Servidor

Neste modelo, o programa cliente não se movimenta, e todas as consultas são feitas aos servidores, via RPC.

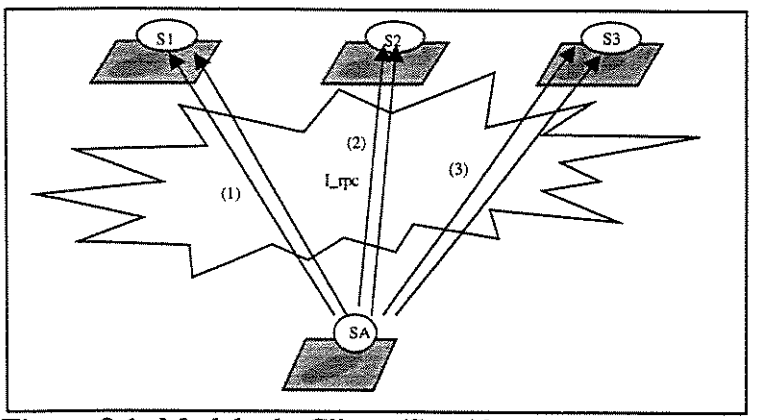

Figura 8.1: Modelo do Cliente/Servidor

O tempo t_{cs} usado pelo programa cliente será:

1) $t_{cs} = \sum_{i=1}^{N} mI_{rpc} = NmI_{rpc}$

Onde:

N= número de servidores visitados.

m= número de invocações RPC necessárias para realizar a consulta em um servidor (assume-se que o mesmo valor m em todos os servidores).

I_{rpc} = tempo de invocação remota com transferência da estrutura de dados em uma rede de grande porte (Internet). Assume-se que será a mesma em todas as invocações.

8.1.2. Modelo do Agente Móvel sem negociação

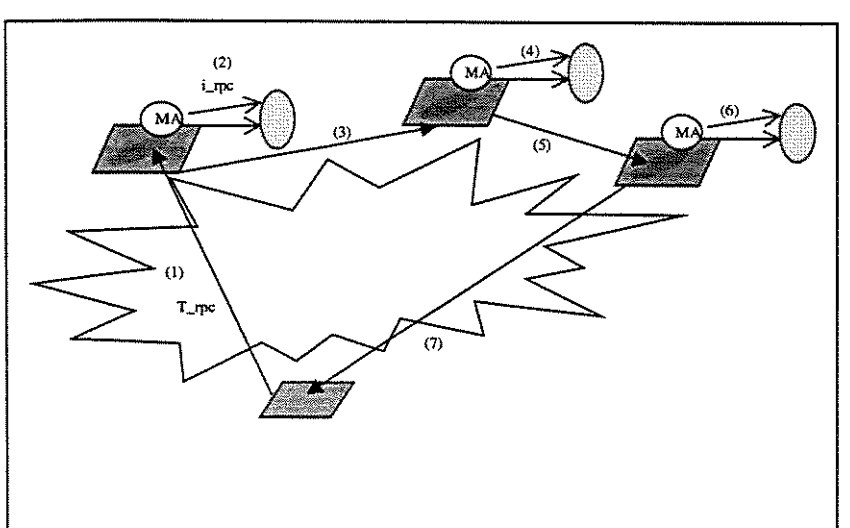

Figura 8.2: Modelo do Agente Móvel sem negociação

Neste modelo, o cliente movimenta-se entre os servidores, como um agente móvel, indo a cada lugar sem consulta prévia para recepção. Portanto, esta estratégia está sujeita a situações onde o agente não será atendido mas, por simplicidade, vamos supor que o agente será sempre recebido em todos os servidores.

Considerando o anterior, pode-se calcular o tempo (t_{am}) que demora o agente para fazer sua pesquisa utilizando a seguinte relação:

2) $$t_{am} = \sum_{i=1}^{N}\sum_{j=1}^{m} i_{rpc} + \sum_{i=1}^{N+1} T_{rpc} = Nmi_{rpc} + (N+1)T_{rpc}$$

Onde:

i_{rpc} = tempo de invocação remota com transferência de estruturas em uma rede local de alta velocidade (intranet).

T_{rpc} = tempo de invocação remota com transferência de octetos em uma rede de grande porte (Internet).

Ou seja, o tempo que demora o agente para fazer sua pesquisa, é composto pelo tempo ocupado em viajar $(N+1)T_{rpc}$ e o tempo gasto em consultar os servidores em cada lugar visitado Nmi_{rpc}.

8.1.3. Modelo do agente móvel com negociação

Neste modelo, o cliente movimenta-se entre os servidores mas, antes de viajar para um novo servidor, negocia seu recebimento e, se aceito, procederá a viajar, de contrário deverá negociar sua viagem com o seguinte servidor de seu itinerário.

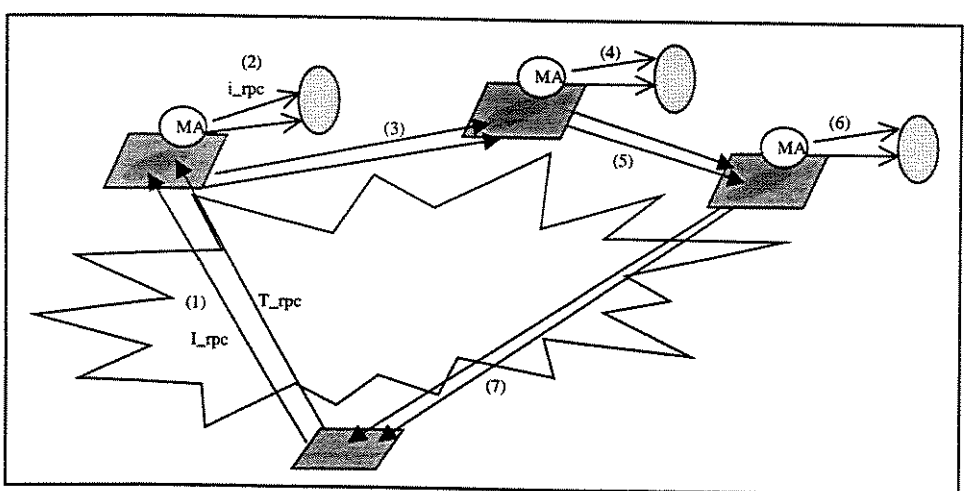

Figura 8.3: Modelo do Agente Móvel com negociação

Se o agente não é aceito em todos os lugares que vai visitar, então o tempo (t_{amn}) que demora o agente móvel para fazer sua pesquisa será descrito pela seguinte fórmula:

$$3)\quad t_{amn} = \sum_{i=1}^{n}(I_{rpc} + T_{rpc} + \sum_{j=1}^{m} i_{rpc}) + \sum_{i=n+1}^{N} I_{rpc} + T_{rpc} = n(I_{rpc} + T_{rpc} + mi_{rpc}) + (N-n)I_{rpc} + T_{rpc}$$

Onde

n = número de negociações com sucesso, n ≤ N

O primeiro termo representa a viagem do agente até aos servidores, após uma negociação com sucesso, o segundo as negociações feitas sem sucesso e o terceiro o regresso do agente até à origem, após término do seu trabalho.

Reordenando:

$$4)\quad t_{amn} = \sum_{i=1}^{n+1} T_{rpc} + \sum_{i=1}^{N} I_{rpc} + \sum_{i=1}^{n}\sum_{j=1}^{m} i_{rpc} = (n+1)T_{rpc} + NI_{rpc} + nmi_{rpc}$$

Assumindo que $i_{rpc} << I_{rpc}$, então:

$$5)\quad t_{amn} = (n+1)T_{rpc} + NI_{rpc}$$

Nesta estratégia, I_{rpc} diz respeito às invocações para negociação e T_{rpc} às invocações para o transporte do agente. As invocações por negociação envolvem a transferências de estruturas de dados, enquanto que as invocações para o transporte do agente envolvem somente a transferência de dados binários. Sob CORBA tem-se demonstrado empiricamente um melhor desempenho na transmissão de octetos que na transmissão de estruturas binárias de dados [Schmidt97]: o desempenho na transmissão de octetos é inferior à transmissão via TCP/IP. Alguns pesquisadores [Strasser97] definiram modelos de desempenho que comparam o uso de RMI e migração de agente, concluindo que, quando se trata do transporte de pouco código, é melhor o uso de RMI, caso contrário, a migração de código.

8.1.4. Comparação dos modelos do Agente Móvel e do C/S

Para fazer a comparação entre estes modelos será necessário impor algumas restrições para facilitar a análise, sem comprometer a consistência das conclusões.

Por isso, vai-se assumir que os tempos (I_{rpc}) das invocações por negociação e os tempos (i_{rpc}) das invocações remotas, com transferência de estruturas em uma rede local de alta velocidade, são constantes e que o número (m) de invocações RPC num servidor não varia.

Logo, para que o modelo do Agente Móvel tenha um melhor desempenho que o modelo do C/S, deve-se cumprir a relação:

6) $t_{am} \leq t_{cs}$

A partir das equações 1) e 2), determina-se que o tempo usado pelo agente para ser transferido até o destino (T_{rpc}), no qual o modelo do agente móvel apresenta um melhor desempenho que do C/S, será descrito pela relação:

7) $T_{rpc} \leq \frac{Nm(I_{rpc} - i_{rpc})}{N+1}$

Se o tempo das invocações remotas, que acontecem nas redes de grande porte como a Internet, são maiores que os tempos das invocações remotas que acontecem nas redes locais, como as intranets, normalmente de alto desempenho, então é possível dizer-se que:

$I_{rpc} >> i_{rpc}$

Logo, a expressão para T_{rpc} fica como:

8) $T_{rpc} \leq \frac{Nm}{N+1} I_{rpc}$

Da equação 8) determina-se que $t_{am} = t_{cs}$ quando:

9) $T_{rpc} = \frac{Nm}{N+1} I_{rpc}$,

o qual será o ponto de interseção das retas desenhadas pelas equações 1) e 2), ver fig. 8.4.

Por outro lado, como I_{rpc}, N e m são constantes, então t_{cs} também será constante.

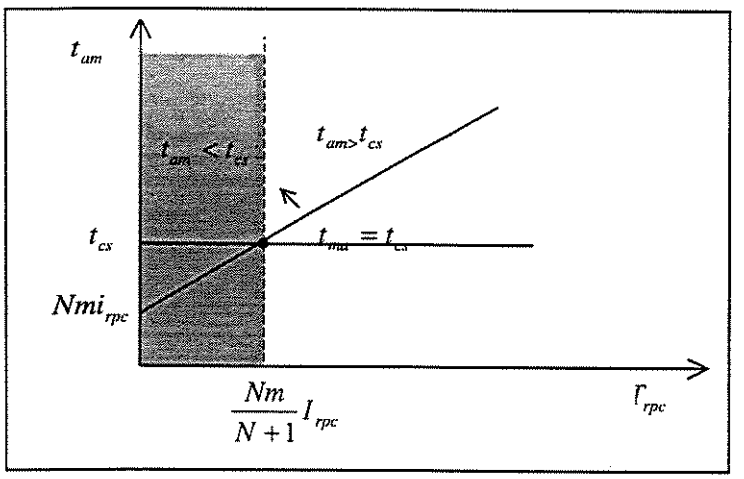

Figura 8.4: Relacionamento entre t_{am} e t_{cs}.

Portanto, estas relações podem-se descrever da seguinte forma:

Para N muito grande (quando visitar muitos servidores) a equação 8) fica como:

10) $T_{rpc} \leq mI_{rpc} \mid_{N \to \infty}$

ou seja, quando uma aplicação demora menos a enviar um agente até o servidor que a realizar as consultas ao servidor através da Internet, então é melhor usar o modelo do agente móvel. Mas isto depende de vários fatores difíceis de quantificar (número de invocações, número de servidores visitados) e de previsões de valores (tempos de transferência e tempos das invocações) que dependem do desempenho da rede (largura da banda, horário, etc.).

8.2. Comparação experimental dos modelos

Com o objetivo de verificar a análise teórica e sua conclusão, fizeram-se testes para comparar os tempos de atraso nas invocações RPC, de uma aplicação baseada no modelo cliente/servidor, com os atrasos na transferência de arquivos utilizando IIOP. Os testes foram realizados na Internet e em uma intranet. Além disso, fizeram-se medições na transferência de arquivos, via Sockets.

8.2.1 Considerações gerais

Para realizar os testes com medições que possam ser comparáveis, fizeram-se as seguintes considerações:

(a) O ambiente dos testes foi a Internet e uma intranet;

(b) Todos os programas foram realizados utilizando Java como linguagem de programação, utilizando-se JDK versão 1.2.1;

(c) A interface da aplicação C/S foi especificada em IDL;

(d) Implementou-se a interface IDL proposta nesta tese, para a transferência de arquivos (transporte IIOP);

(e) Utilizou-se o "Visibroker for Java" versão 4.0 beta, como ferramenta CORBA para implementar as interfaces IDL das aplicações C/S e do transporte IIOP;

(f) Para referência, mediu-se o desempenho na transferência de arquivos, utilizando *Sockets* do tipo *Stream*.

Descrição do ambiente dos testes

Quando os testes foram feitos na Internet, participaram: um computador pessoal (Pentium II), que normalmente hospedou o processo cliente, localizado no laboratório da Escola de Computação, da Universidade de Magalhães (UMAG), em Punta Arenas, Chile. Na cidade de Campinas-SP, Brasil, no Laboratório do Departamento de Engenharia de Computação e Automação Industrial, da Universidade de Campinas (UNICAMP), escolheu-se uma estação de trabalho Sun para conter o processo servidor.

No caso dos testes feitos em uma intranet, estes foram realizados numa rede local Ethernet (10 Mbps) do laboratório da Escola de Computação, da Universidade de Magalhães (UMAG), em Punta Arenas, Chile e os participantes foram: um computador pessoal Pentium II sob Windows 95, como servidor, e um computador pessoal Pentium II sob Linux 6.1, como cliente.

Descrição das aplicações envolvidas nos testes

A aplicação C/S simula uma invocação simples de um servidor.

A aplicação simula a compra de um produto qualquer. Para isso, o cliente consulta primeiro o produto e posteriormente compra. Antes disso, o fabricante de objetos que suporta a interface "FabricaLoja" é contatado, para obter a referência de um objeto do tipo "Loja", sobre o qual serão feitas as consultas e compras.

A definição da interface do servidor é a seguinte:

```
// CompraVendas.idl

module CompraVendas {

      typedef float valor;

      interface Loja {
            valor consulta (in string nome_prod);
            boolean compra (in string nome, in short quantidade);
      };

      interface FabricaLoja {
            Loja cria_Loja();
        };

};
```

O protocolo de transporte IIOP para o transporte de arquivos foi apresentado e descrito no capítulo anterior, apresentando-se aqui novamente sua especificação IDL para maior clareza da descrição dos testes.

```
// Transport.idl
module Transport {

        typedef sequence< octet > BinStream;

        interface IiopTransport {
                boolean begin_tx();
                boolean tx_agent(in BinStream data, in short lenght);
                boolean end_tx();
        };
        interface TransportFactory {
                IiopTransport create_IIOP_transport();
        };
};
```

Da especificação pode-se observar que a interface "IiopTransport" possui três operações que permitem a transferência de arquivos binários. A interface "TransportFactory" permite fabricar e devolver a referência de um objeto CORBA do tipo "IiopTransport".

Como atualmente os objetos CORBA não podem ser ainda encontrados através da Internet por seu nome, de forma transparente à aplicação (a primeira vez que vai ser usado), sua referência deve se converter em *string* e armazenada em um arquivo, a partir do qual o cliente poderá obtê-la por meios externos e fazer o processo inverso. No caso deste teste, o arquivo foi transmitido manualmente, via FTP, para o lado cliente.

8.2.2 Resultados obtidos

Como se explicou anteriormente, fizeram-se testes tanto na Internet como em uma intranet. Mediram-se os tempos que um cliente demora em ser atendido pelo servidor, para diferentes números de consultas, e o tempo que demora o transporte, via IIOP e via Socket, para transferir arquivos de diferentes tamanhos. Todas as medições foram repetidas entre 30 e 50 vezes.

Dos valores experimentais obtidos, calculou-se a média aritmética e utilizando uma ferramenta matemática (KURV for Windows versão 1.1), se escolheram as equações que apresentavam a melhor correlação com os valores médios obtidos empiricamente para os diferentes cenários. Apartir das equações escolhidas se procedeu a gerar as tabelas de dados apresentadas nas Tabelas 8.1 e 8.2.

No caso dos tempos de transporte apresentam-se, na Tabela 8.1, os valores obtidos nos testes realizados na Internet e na intranet.

Tamanho do arquivo (bytes)	Internet		Intranet	
	Transporte IIOP (segundos)	Transporte Sockets (segundos)	Transporte IIOP (segundos)	Transporte Sockets (miliseg.)
1,050	21.16	0.33	4.2	4.55
2,039	22.57	0.65	4.25	8.83
4,412	25.97	1.41	4.36	19.07
7,202	29.95	2.31	4.5	31.06
(1) 10,216	34.25	3.29	4.65	43.94
(2) 11,448	36.01	3.69	4.71	49.19
(3) 12,286	37.21	3.96	4.75	52.76
16,007	42.51	5.19	4.94	68.52
(4) 22,653	51.98	7.40	5.27	96.43
26,918	58.04	8.84	5.48	114.18
38,048	73.84	12.67	6.03	159.87
55,093	89.95	18.73	6.87	228.14
78,597	131.04	27.48	8.04	318.87
99,422	160.18	35.61	9.07	395.96
108,460	172.79	39.25	9.52	428.46
145,206	223.74	54.74	11.34	554.55
207,825	309.47	83.68	14.44	747.20
257,516	376.53	108.93	16.9	880.14
320,687	460.54	143.95	20.03	1023.66
479,037	665.00	246.13	27.87	1258.11

Tabela 8.1: Valores obtidos para o transporte de arquivos via IIOP e Socket, para os testes feitos na Internet e na intranet.

(1) Corresponde ao arquivo que contém o programa fonte (.java) do lado cliente e do lado servidor da aplicação C/S, comprimido em formato zip.

(2) Corresponde ao arquivo que contém o programa fonte (.java) do transporte IIOP, comprimido em formato zip.

(3) Corresponde ao arquivo que contém o programa executável (.class) do lado cliente e do lado servidor da aplicação C/S, comprimido em formato zip.

(4) Corresponde ao arquivo que contém o programa fonte (.java) e executável (.class) do lado cliente e do lado servidor da aplicação C/S, comprimido em formato zip.

Para a aplicação C/S obtiveram-se os valores que se apresentam na Tabela 8.2. Nela, a coluna de "Consultas", indica as vezes que a operação "compra" foi invocada e as outras colunas, expressam o tempo que o cliente demorou em fazer as consultas.

Consultas	Cliente na Internet (segundos)	Cliente na intranet (segundos)
1	13.66	4.076
2	16.49	4.065
3	19.60	4.075
4	22.78	4.088
5	25.99	4.102
6	29.22	4.118
7	32.46	4.133
8	35.72	4.150
9	38.98	4.166
10	42.24	4.182
11	45.52	4.199
12	48.79	4.216
13	52.07	4.233
14	55.36	4.250
15	58.65	4.268
16	61.94	4.285
17	65.24	4.303

Tabela 8.2: Tempos de demora da aplicação C/S através da Internet e da intranet.

É importante salientar que, sob CORBA na Internet, o cliente demora cerca de 8 segundos para estabelecer a comunicação com o servidor: este valor está embutido nas medições realizadas. O valor médio das invocações foi de 2,8 segundos. No caso da intranet, o cliente demora de 4 a 4,5 segundos para estabelecer a conexão com o servidor, e cada invocação demora de 20 a 100 milisegundos.

8.2.3 Discussão dos resultados

Para discutir os resultados obtidos dos testes, primeiro vai-se analisar o comportamento do protocolo IIOP e posteriormente o comportamento da aplicação C/S, ambos considerando os ambientes da Internet e de uma intranet. Em seguida comparam-se os tempos de

transportes via IIOP e Sockets. Finalmente comparam-se os modelos do agente móvel e de C/S, considerando a análise anterior dos resultados obtidos nos testes.

Comportamento do transporte IIOP

Como era de esperar (ver fig. 8.5), os tempos na transferência de arquivos entre valores de

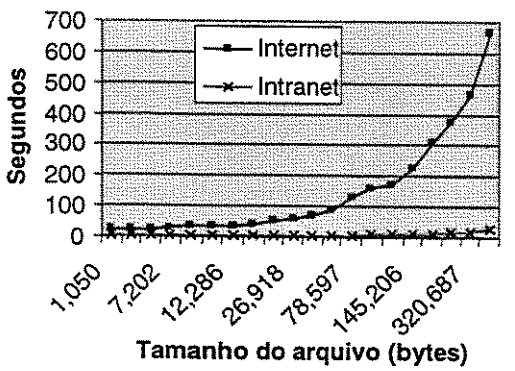

Figura 8.5: Demora na transferencia vía IIOP

1K e 467Kbytes, são maiores na Internet.

Na figura 8.6, apresenta-se o comportamento em termos de desempenho, podendo-se observar que, na Internet a variação da velocidade de transmissão (49 à 720 bytes/seg) é menor que na intranet (250 à 17.188 bytes/seg).

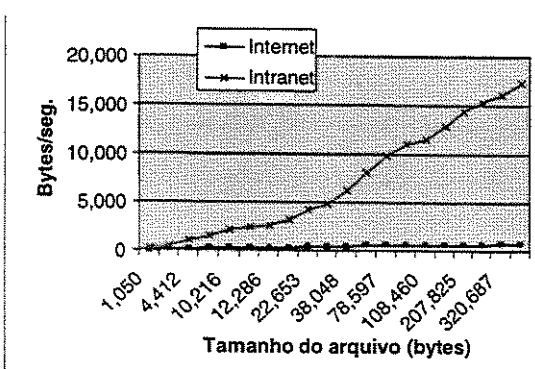

Figura 8.6: Velocidade transferencia de arquivos vía IIOP

Comparando ambos os desempenhos (ver figura 8.7), a transferência via IIOP numa

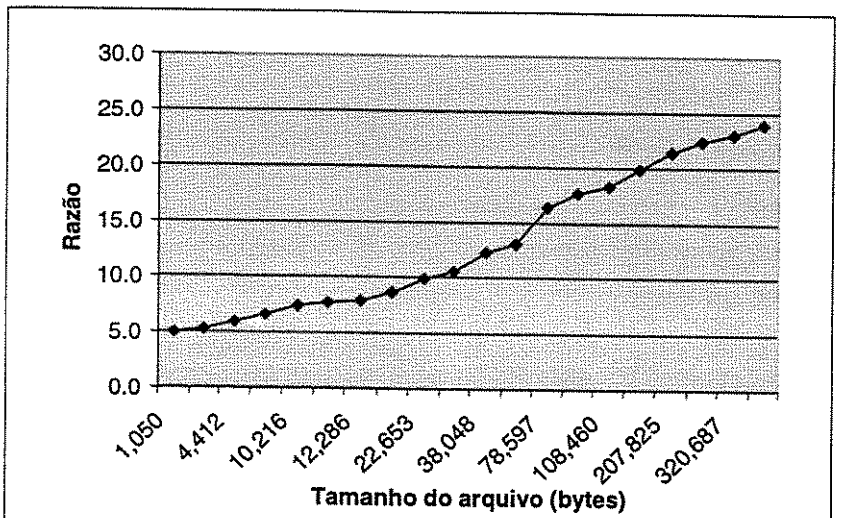

Figura 8.7: Razão de transferencia de arquivos intranet/Internet vía IIOP

intranet é 5 a 24 vezes mais rápida que na Internet, proporcional ao tamanho do arquivo.

Comportamento da aplicação C/S

Como se observa na figura 8.8 e como era de esperar, confirma-se que a aplicação C/S é mais lenta na Internet.

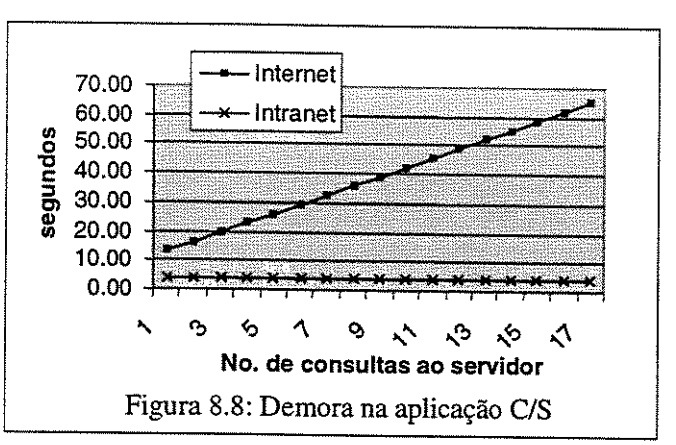

Figura 8.8: Demora na aplicação C/S

Na figura 8.9, apresenta-se o comportamento só na intranet, para destacar a pouca

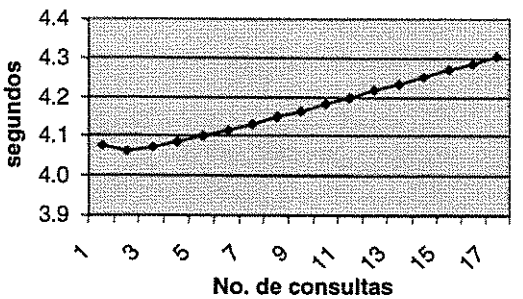

Figura 8.9: Demora da aplicação C/S na intranet

variabilidade para diferentes números de consultas. Ou seja, o tráfego na Internet tem um impacto negativo maior no desempenho da aplicação C/S que o tráfego na intranet. Comparando ambos os comportamentos, pode-se dizer que a aplicação C/S é de 3 a 15 vezes mais lenta na Internet que na intranet, como se observa na figura 8.10.

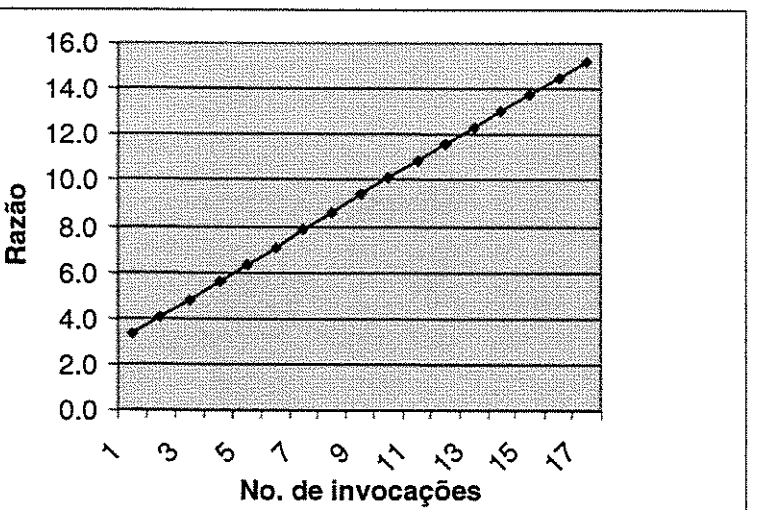

Figura 8.10: Razão Internet/intranet na demora aplicação C/S

IIOP versus Socket

A implementação da transferência de arquivos, utilizando Sockets, serviu para comparar seu desempenho com o transporte IIOP. As medições feitas mostraram que, para tamanhos

de arquivos relativamente pequenos, o desempenho dos Sockets é muito superior, com maior destaque na intranet (ver figuras 8.11 e 8.12).

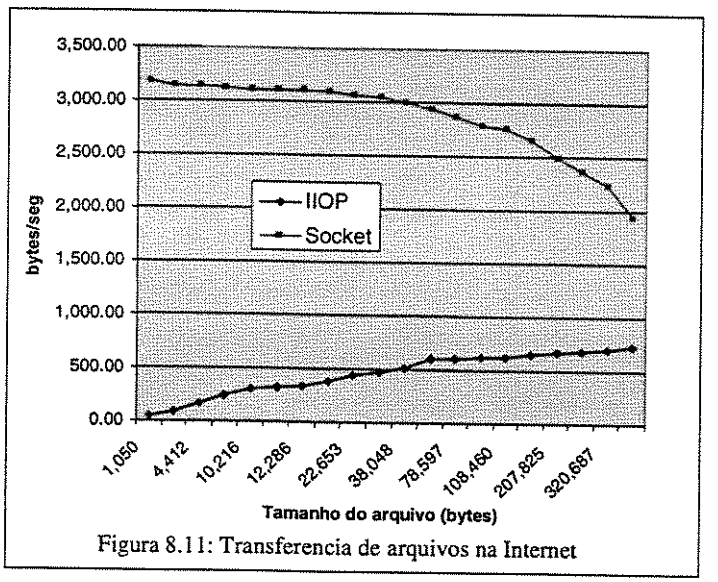

Figura 8.11: Transferencia de arquivos na Internet

À medida que o tamanho do arquivo aumenta, a razão do desempenho diminui, sendo os

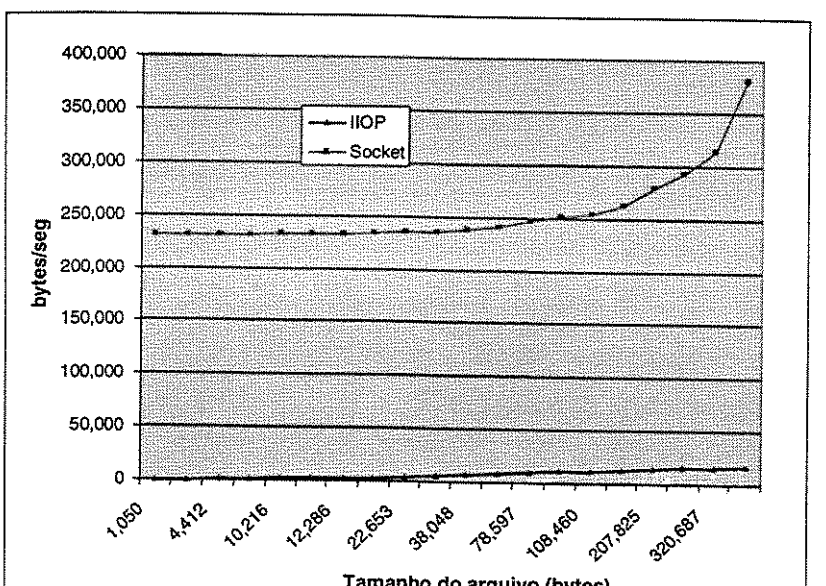

Figura 8.12: Transferencia de arquivos na intranet

valores comparáveis aos da Internet, na qual o desempenho dos Sockets tende a diminuir

para 2 vezes mais rápido que IIOP. Em geral, a variação da razão do desempenho Socket/IIOP é menor no caso da Internet, como observa-se na figura 8.13.

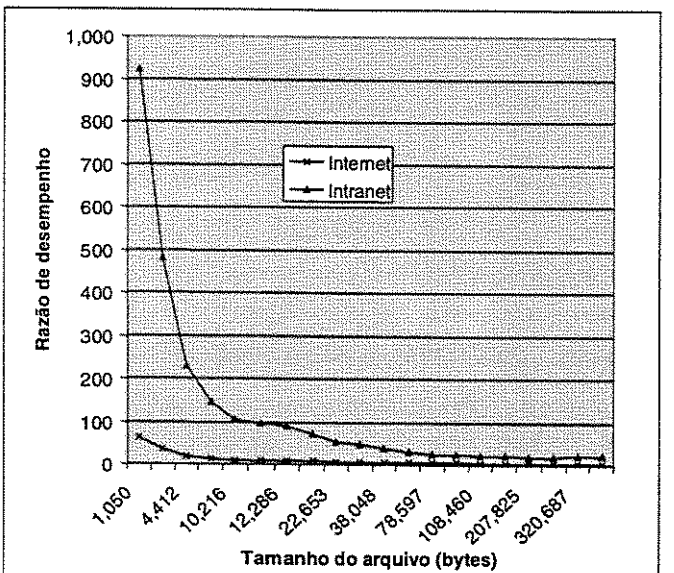

Figura 8.13: Comparação razão de desempenho Socket/IIOP

Comparação do modelo do agente móvel com o modelo do C/S

Da análise teórica concluiu-se que é necessário que o tempo para transferir o agente seja

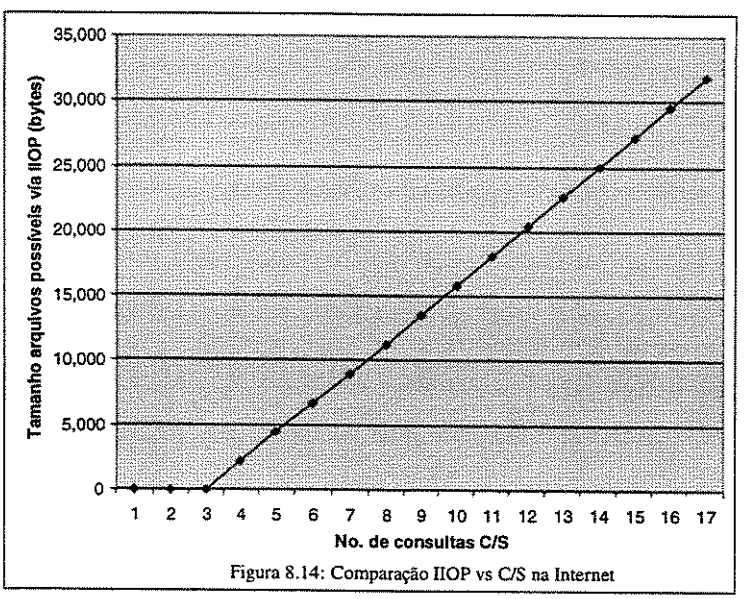

Figura 8.14: Comparação IIOP vs C/S na Internet

menor que o tempo total ocupado pelo cliente para consultar um servidor, para que o modelo do agente móvel seja mais adequado que o modelo do C/S.

A partir das medições realizadas, procurou-se a forma de avaliar esta condição, em termos quantitativos. Para isso, calculou-se o tamanho dos arquivos possíveis de serem transmitidos, referentes aos tempos usados pela aplicação C/S para realizar o seu trabalho, para diferentes números de consultas. O cálculo foi feito considerando o desempenho que o transporte IIOP teve na rede correspondente, cujos resultados se apresentam na figura 8.14 para a rede Internet e na figura 8.15 para uma rede intranet.

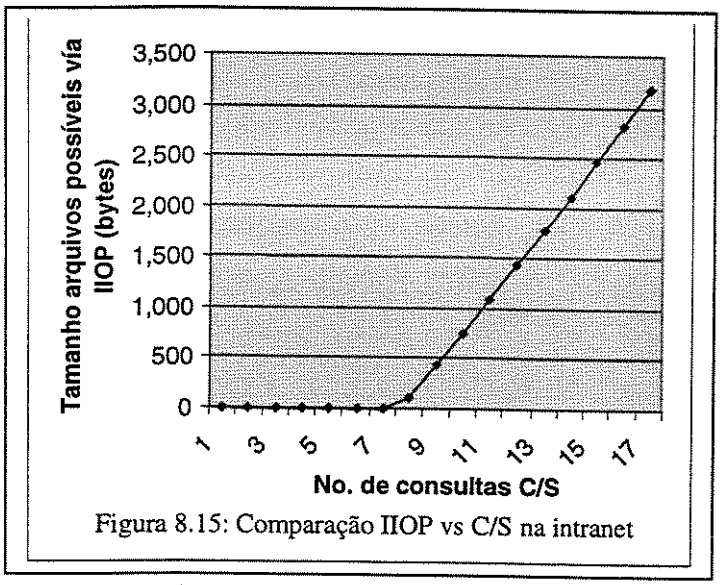

Figura 8.15: Comparação IIOP vs C/S na intranet

Das figuras, pode-se observar que no caso de tratar-se de uma interação C/S com 17 consultas, o transporte IIOP permitiria transmitir agentes de até aproximadamente 30Kbytes na Internet e até aproximadamente 3Kbytes na intranet. Tamanhos maiores tornam o modelo C/S mais adequado. No caso da Internet, o comportamento apresenta uma tendência a permitir transferir agentes de tamanhos maiores, à medida que o tempo de interação entre o cliente e o servidor aumenta. Mas isso não acontece na intranet, na qual se observa um comportamento errático, uma vez que os tempos de interação da aplicação C/S apresentam pouca variação para diferentes números de consultas, ou seja, a demora de uma interação na intranet é quase independente do número de invocações feitas (figura 8.9). De fato, calculou-se que mais de 90 % do tempo de interação corresponde ao tempo que uma invocação CORBA demora em estabelecer a conexão com o servidor, ou seja, os tempos na interação C/S dependem mais do tempo necessário para se estabelecer uma conexão que do número de invocações realizadas. O mesmo não acontece na Internet, onde se observou que o tempo de invocação é cerca de 8 segundos, de 70 à 30 % do total, à medida que o número de consultas aumenta.

Em resumo, a rede Internet apresenta um cenário onde o modelo do agente móvel, utilizando IIOP como transporte, só é vantajoso sob certas condições, que no caso dos testes, resultou ser para agentes cujos tamanhos não sejam superiores a 30Kbytes. Já na rede intranet o transporte de agentes via IIOP não é recomendável.

Este cenário muda totalmente quando se usam os Sockets como via de transporte para os agentes. Para o caso da Internet, o tamanho do agente quintuplica-se em relação valor anteriormente indicado. No caso destes testes, seria possível transferir agentes de tamanhos de até 150Kbytes.

Na intranet, e para os testes realizados, onde se consideraram tamanhos de arquivos de até 467Kbytes, a transferência via Sockets é da ordem de grandeza de milisegundos, enquanto que a interação C/S apresenta resultados da ordem de grandeza de segundos (4 a 7 segundos). Por exemplo, na transferência de arquivos, via Sockets, para tamanhos entre

1,02 Kbyte e 467 Kbyte, a demora foi entre 5 e 1258 milisegundos (ver Tabela 8.1), enquanto que no modelo C/S esses valores ficaram entorno de 4 segundos (ver Tabela 8.2).

Isso indica que sempre seria vantajoso aplicar o modelo do agente móvel nas redes intranet, quando Sockets são usados como meio de transporte de agentes. Na Internet, haverá um tamanho do agente, a partir do qual já não será vantajoso o modelo do agente móvel.

8.3. Resumo

A partir da análise teórica dos modelos do agente móvel e do C/S, determinou-se que o modelo do agente móvel será vantajoso se o transporte do agente demorar menos tempo que o necessário para o cliente fazer suas consultas no servidor.

Fizeram-se os testes para comparar, em termos quantitativos, ambos os modelos, considerando-se tanto a Internet como a intranet, como meio de comunicação. A partir das medições feitas, determinaram-se os tamanhos máximos dos agentes. Sem dúvida que, para um mesmo meio de comunicação, estes valores mudam fortemente, dependendo do horário e das variáveis de tráfego na rede.

No caso de IIOP como meio de transporte, observou-se que na Internet é possível encontrar-se tamanhos de agentes que resultem em vantagem, o mesmo já não foi possível no caso da intranet.

Com o uso dos Sockets como meio de transporte de agentes, observou-se que, tanto na Internet como na intranet, sempre será vantajoso o modelo do agente móvel. No primeiro caso, obtiveram-se tamanhos de agentes cinco vezes superiores que no caso de IIOP, como meio de transporte de agentes. No segundo caso, mostrou-se que o modelo do agente sempre era melhor que o modelo do C/S, ou seja, que na intranet também é factível o uso o modelo de agentes móveis em trabalhos onde se deve consultar servidores seqüencialmente.

No transporte dos agentes utilizando diretamente Sockets, resultou ser mais eficiente que IIOP. Mas pode-se considerar uma desvantagem o fato de que o protocolo de transporte de agentes será de difícil manutenção. IIOP é mais flexível, já que permite sua publicação na rede (interfaces IDL). Com certeza, o protocolo de transporte poderá sofrer modificações posteriormente, para adaptar-se às novas características que a rede Internet irá tendo no futuro. O fato de serem usadas especificações em IDL, publicadas nos Repositórios de Interfaces IDL, facilita sua rápida utilização, inclusive fazendo-se uso dos mecanismos de invocação dinâmica (DII). Todas estas ferramentas não estão disponíveis a nível de Sockets.

9. Conclusões finais e trabalho futuro

9.1. Conclusões finais

Durante a etapa de busca da informação sobre os problemas do agente móvel, concluiu-se que estes problemas não estão sendo tratados publicamente, utilizando-se protocolos de transporte já existentes sem outras justificativas. Muitos dos projetos estão sendo desenvolvidos num escopo particular onde o objetivo fundamental é ganhar experiência no desenvolvimento de aplicações baseadas em agentes móveis, sem preocupação com o desempenho da plataforma na rede.

Estima-se que o presente trabalho contribui para formar uma opinião sobre os problemas que ocorrem com o transporte de agentes, em um ambiente aberto, distribuído e heterogêneo. No trabalho teve-se como objetivo determinar uma arquitetura de transporte e analisar seu desempenho.

Neste trabalho, a análise dos problemas associados ao transporte de agentes e às funcionalidades de uma arquitetura, foram feitas de forma independente da tecnologia para implementar o STA.

Na proposta, estabeleceu-se um modelo de transporte que opera em duas etapas: negociação e posterior transferência do agente, se a negociação tem sucesso.

A característica de negociação só foi encontrada no projeto MOA, mas sem outras informações sobre a sua implementação. Esta característica é importante, uma vez que poderá tornar-se crítica, à medida que o ambiente de agentes ou das aplicações cresça também em complexidade.

Dos testes pode-se concluir que é mais vantajoso negociar usando invocações CORBA, que enviando um pacote ao destino com os termos da negociação, porque, sua demora não tem um impacto negativo no desempenho final do processo de transporte do agente. Por outro lado, estar-se-á usando um mecanismo distribuído que deixará à plataforma preocupar-se em contatar a agência destino.

Entretanto, a etapa de transferência caracteriza-se pela transmissão demorada de dados binários de comprimento maior, na qual o processo do marshalling não é considerado; logo, em princípio, é melhor usar protocolos de transporte mais adequados que usem diretamente as interfaces Sockets. Os testes demonstraram, mesmo assim, que IIOP também pode ser uma alternativa válida na transmissão de informação binária.

Na hipótese teórica inicial, considerou-se a Internet, como sendo o principal ambiente para usar o modelo de agentes móveis, com vantagens sobre o modelo de C/S. Os testes demonstraram que o modelo de agente móvel também pode ser adequado nas redes intranet, sempre que o protocolo de transporte utilizado seja de melhor desempenho que o IIOP, como aconteceu ao transferir os agentes via Sockets.

A partir dos testes foi também possível determinar tamanhos de agentes que permitiriam um melhor desempenho do modelo de agente móvel sobre o modelo de C/S, usando IIOP como protocolo de transporte e um protocolo simples para a transferência de arquivos vía Sockets.

A utilização de CORBA para implementar um ambiente para os agentes móveis justifica-se porque satisfaz os principais requerimentos destes ambientes, como: heterogeneidade, escalabilidade, integração de diferentes tecnologias de software e complexidade crescente dos serviços na rede.

O trabalho desenvolvido permite mostrar que, o modelo de agente móvel tem potencialidades reais que podem ser aproveitadas no desenvolvimento de novas tecnologias de software, principalmente nas aplicações em redes.

Atualmente está em implantação nos Estados Unidos o projeto da Internet2 [Wiseman99], para pesquisar novas tecnologias de hardware e software para implementar uma rede Internet de alto desempenho, esperando-se chegar a resultados 10 a 100 vezes mais rápidos de operação que na rede Internet atual. . Considerando esta nova realidade, alguns autores [Waldo97] acreditam que, quando a rede Internet melhorar seu desempenho com as novas tecnologias, a vantagem que o modelo de agente móvel eventualmente possa ter será irrelevante. Mas, em nossa opinião, acredita-se que o desempenho melhorará para todo tipo de aplicações na rede, incluindo aquelas baseadas no agente móvel, ampliando ainda mais as vantagens do modelo de agente móvel sobre os modelos atuais.

Por outro lado, os testes na intranet, usando Sockets para a transferência de agentes, demonstraram que é factível transferir agentes de tamanhos muito maiores que na Internet, melhorando ainda mais a vantagem deste modelo com relação ao modelo de C/S. Se se considerar a intranet similar à futura rede Internet de alto desempenho, poderá afirmar-se que o modelo de agente móvel sempre poderá apresentará vantagens reais sobre o modelo de C/S.

É evidente que o transporte de agentes não é a única característica para avaliar o desempenho do modelo de agente móvel, principalmente com relação ao modelo de C/S. Outras características deverão ser consideradas, como: estratégias de transferência, modelos

de programação que demonstrem a afinidade do modelo com certos tipos de aplicações e facilidades de desenvolvimento em relação a outros modelos existentes.

9.2. Trabalhos futuros

Tendo implementado o Serviço de Transporte de Agentes, será interessante testar o desempenho de agentes baseado nas estratégias de movimento. Por exemplo poderá comparar-se o desempenho quando o agente entrega à agência o endereço do próximo lugar a visitar ou o itinerário, para que a agência possa escolher automaticamente o endereço seguinte, no caso de não haver sucesso na negociação.

Por outro lado se a agência deve gerenciar o ciclo de vida do agente considerando ambientes heterogêneos, então, a definição de uma estrutura básica padrão de um agente parece ser essencial. Neste sentido, é interessante analisar-se o modelo de Componentes da OMG como ponto de partida para desenvolver essa estrutura.

Desenvolver uma aplicação que opere de forma seqüencial, utilizando este serviço, para assim avaliar a flexibilidade que oferece o modelo do agente móvel na programação de aplicações baseadas neste paradigma.

Na presente proposta o código do agente viaja junto com o agente quando visita cada servidor. Seria interessante avaliar a alternativa de transportar o código desde um servidor associado ao agente. Assim, o agente teria a possibilidade de se adaptar às condições do ambiente no destino. Desta forma, pode-se incorporar mais flexibilidade ao mecanismo de transporte do serviço.

Aprofundar mais a questão da etapa de negociação do agente para abordar situações mais complexas.

10. Referências

[AdAstra99] "Jumping Beans", White Paper, 11 Outubro 1999, Ad Astra Enginnering Inc. http://www.Jumpingbeans.com/

[Aglet97] Mitsuru Oshima and Guenter Karjoth: "Aglets Specification (1.1)", IBM, 20 Maio 1997. http://www.trl.ibm.co.jp/aglets/spec11.html

[Atp97] Danny B. Lang and Yariv Aridor: "Agent Transfer Protocol – ATP/0.1", IBM Tokyo Research Laboratory. http://www.trl.ibm.co.jp/atp/atp.htm

[Bagrodia95] R. Bagrodia, W.W. Chu, L. Kleinrock, and G. Popek, 'Vision, Issues, and Architecture for Nomadic Computing', IEEE Personal Communication, Dezembro 1995.

[Baldi98] Mario Baldi and Gian Pietro Picco: "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications". 20th International Conference on Software Engineering (ICSE '98), Kyoto, Japan, Abril 1998.

[Baumann97] J. Baumann, and N. Radouniklis: "Agent Groups in Mobile Agent Systems", DAIS'97, in "Distributed Applications and Interoperable Systems", Chapman & Hall

[Bic97] Lubomir F. Bic, Michael B. Dillencourt, and Munehiro Fukuda:"Mobile Agents, DSM, Coordination, and Self-Migrating Threads: A Common Framework", 19 Junho, 1997. http://www.ics.uci.edu/~bic/messengers/papers/compare.ps

[Bic99] L. F. Bic, M. B. Dillencourt, M. Fukuda: "Mobile Network Objects", Encyclopedia of Electrical and Electronics Enginnering, Vol. 13 (J. Webster, Ed.), John Wiley & Sons, Inc., New York, 1999.

[Bieszczad98a] Andrzej Bieszczad, Syed K. Raza, Bernard Pagurek, and T. White: "Agent-Based Schemes for Plug-and-Play Networ Components". In Intelligent Agents for Telecommunications Applications, Albaryak, S. and Garijo, F. J. (Eds.), Springer-Verlag, Berlim, 1998.

[Bieszczad98b] Andrzej Bieszczad and Bernard Pagurek: "Network Management Application-Oriented Taxonomy of Mobile Code", Proceedings of the EEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, 15-20 Fevereiro, 1998.

[Bradshaw96] Jeffrey M. Bradshaw: "KaoS: An Open Agent Architecture Supporting Reuse, Interoperability, and Extensibility". http://ksi.cpsc.ucalgary.ca/KAW/KAW96/bradshaw/KAW.html

[Breugst98] Markus Breugst, Lars Hagen, and Thomas Magedanz: "Impacts of Mobile Agent Technology on Mobile Communications System Evolution", IEEE Personal Communication Magazine, Agosto 1998

[CACM94] Communications of the ACM Journal, Intelligent Agents, 37(7), Julho

1994.

[Cai96] T. Cai, P. Gloor, and S. Nog: "DartFlow: A Workflow Management System on the Web using Transportable Agents", http://www.cs.dartmouth.edu/reports/abstracts/TR96-283

[CoABS00] "Control of Agent Based Systems". Proyecto apoiado pelo DARPA/USA. http://coabs.globalinfotek.com/

[Compon99] "CORBA Components", Joint Revised Submission, por BEA, CRCDST, Expersoft, IBM, Inprise, IONA, Oracle, Rogue Wave y Unisys, OMG TC doc. orbos/99-04-16, Junho 1999.

[CORBA99] CORBA – IIOP 2.3 specification, Formal/99-10-07 http://www.omg.org/technology/documents/formal/corba_2.htm

[Cugola96] Gianpaolo Cugola, Carlo Ghezzi, Gian Pietro Picco, and Giovanni Vigna: "A Characterization of Mobility and State Distribution in Mobile Code Languages". In Proceedings of the 2nd ECOOP Workshop on Mobile Objects, 8-9 Julho 1996, Linz, Austria, dpunkt.verlag.

[Cullens95] "Serialization and MFC: Extending MFC for cross-platform portability", Dr. Dobb's Journal, #229 Abril 1995

[Chang96] Daniel T. Chang, and Danny B. Lange: "Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW", OOPSLA '96 Workshop

[Chess95] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, 'Itinerant Agents for Mobile Computing', IEEE Personal Communication Magazine, 2(5), Outubro 1995.

[Chess97] D. Chess, C. Harrison, and A. Kershenbaum: "Mobile Agents: Are they a good idea", Lecture Notes in Computer Science 1222, Springer-Verlag, 1997.

[Dale97] Jonathan Dale, and David C. DeRoure: "A Mobile Agent Architecture for Distributed Information Management". http://wwwcosm.ecs.soton.ac.uk/publications/papers/Voyager/papers/ieeeic97.htm

[EJB99] Enterprise JavaBeans Specification, v1.1, Sun Microsystems Inc.

http://www.javasoft.com/products/ejb/newspec.html

[Etzioni95] Oren Etzioni and Daniel S. Weld: "Intelligent Agents on the Internet: Fact, Fiction, and Forecast", IEEE Expert, Agosto 1995

[Event97] CORBA Services: Event Management Service

ftp://ftp.omg.org/pub/docs/formal/97-02-09

[Externaliz97] CORBA Services: Externalization Service ftp://ftp.omg.org/pub/docs/formal/97-02-13

[Franklin96] Stan Franklin, and Art Graesser: "Is it an Agent, or just a Program?: A

Taxonomy for Autonmous Agents".
http://www.msci.memphis.edu/~franklin/AgentProg.html

[Fuggetta98] A. Fuggetta, G. P. Picco, and G. Vigna: "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol. 24, No. 5, Maio 1998

[Global00] "The CoABS Grid", Global Info Tek, Inc. Março, 2000

http://coabs.globalinfotek.com/coabs_public/coabs_pdf/gridvision.pdf

[Gollmann98] Dieter Gollmann and Ulrich Lang: "Security in CORBA based Electronic Commerce Systems". Elsevier Science, Information Security Technical Report, vol. 3, no. 2, pp. 64–70, 1998

[Grasshopper99] "Grasshopper Development System Light Edition, Release 1.2: Basics and Concepts", Fevereiro 1999, IKV++ GmbH, Berlin, Alemanha. http://www.ikv.de

[Gray95] Robert Gray: "Agent Tcl: Alpha Release 1.1", 01 Dezembro 1995. http://agent.cs.dartmouth.edu/manual/doc.1.1.ps.gz.

[Gray97] Robert Gray: "Agent Tcl: A flexible and secure mobile-agent system". Ph.D. Thesis, Department of Computer Science, Dartmouth College, Dezembro, 1997. http://actcomm.dartmouth.edu/ papers/gray:thesis.pdf

[Halls97] David Alan Halls: "Applying Mobile Code to Distributed System", tese de doutorado, Junho, 1997.
http://www.crema.unimi.it/mirror/scheme/thesis/

[Harrison95] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum: "Mobile Agents: Are they a good idea?", IBM Research Report, 28 Março, 1995. http://www.nal.ics.es.osaka-ac.jp/~shirai/Violet/rinkou/rinkou0523.html

[Hauck96] Franz J. Hauck, Maarten van Steen, and Andrew S. Tanenbaum: "A location service for Worlwide Distributed Objects", Position Paper for ECOOP '96 Wokshop on Mobility and Replication, 10 Abril, 1996

[Henning98] Michi Henning: "Binding, Migration, and Scalability in CORBA". http://www.ooc.com.au/staff/michi/cacm.pdf

[Hohl97] Fritz Hohl, Peter Klar, and Joachim Baumann: "Efficient Code Migration for Modular Mobile Agents", Setembro 1997.
http://cui.unige.ch/~ecoopws/ws97/papers/hohl.ps.gz

[Hurst97] Leon Hurst, and Pádraig C. F. Somers: "Mobile Agents – Smart Messages" Lecture Notes in Computer Science, No. 1219, Springer Verlag, 1997

[IAG97] Intelligent Agents Group: "Intelligent Agents in Telecommunications". http://www.cs.tcd.ie/research_groups/aig/iag/tereport.html

[Inprise99] Inprise Borland Co. http://www.inprise.com/

[IntNaming98a] Interoperable Naming Service: Joint Initial Submission by IONA and Nortern Telecom, 9 Março 1998,

ftp://ftp.omg.org/pub/docs/orbos/98-03-03.pdf

[IntNaming98b] Interoperable Naming Service: Joint Initial Submission by IBM, Oracle and Sun Microsystems, 21 Dezembro 1998, ftp://ftp.omg.org/pub/docs/orbos/98-12-16.pdf

[Jacobsen99] Kjetil Jacobsen and Dag Johansen:"Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code", Proceedings of the Symposium on Applied Computing (ACM SAC'99), Fevereiro, 1999

[Java96] Java Language Specification, v1.0, por James Gosling, Bill Joy e Guy Steele, Addison-Wesley, Agosto 1996. http://java.sun.com/docs/books/jls/index.html

[JavaBeans99] JavaBeans Specification v1.01 http://www.javasoft.com/beans/docs/spec.html

[JavaSerial98] Java Object Serialization Specification

http://java.sun.com/products/jdk/1.2/docs/guide/serialization/spec/

[JDBC99] Java Data Base Connector, Sun Microsystems Inc.

http://www.javasoft.com/products/jdbc/index.html

[JNDI99] Java Naming Directory Interface, Sun Microsystems Inc.

http://www.javasoft.com/products/jndi/docs.html#12

[Johansen95] Dag Johansen, Robbert van Renesse, and Fred B. Schneider: "Operating System Support for Mobile Agents", Position Paper for 5th IEEE Workshop on Hot Topics in operating Systems, 1995.

[Johansen98] Dag Johansen: "Mobile Agent Applicability", Proceedings of the Mobile Agents, Springer Verlag LNCS, Stuttgart, 9-11 Setembro 1998.

[JTA99] Java Transaction API, Sun Microsystems Inc.

http://www.javasoft.com/products/jta/index.html

[Kiniry97] J. Kinery and D. Zimmerman: "A Hands-On Look at Java Mobile Agents", IEEE Internet, Julho-Agosto, 1997

[Knabe95] Frederick C. Knabe: "Language Support for Mobile Agents", tese de doutorado, Dezembro 1995. http://agents.umbc.edu/papers/knabe.shtml

[Kotz97] David Kotz, Robert Gray, Daniela Rus, Sumit Chawla, and George Cybenko: "Agent Tcl: Targeting the needs of mobile computers". IEEE Internet Computing, 1(4):58-67, Julho/Agosto, 1997

[Kotz99] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko: "Mobile agents for mobile computing". In Dejan Milojicic, Frederick Douglis and Richard Wheeler, editors, Mobility, Mobile Agents and Process Migration - An Edited Collection, 1999.

[Li96] Weiyi Li and David G. Messerschmitt: "Java-to-go: Itinerative

Computing Using Java", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Setembro 1996. http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[Liao96] Willy S. Liao, and Roy H. Campbell: "An Interprocess Communications Design for Migrating Objects", ECOOP-96, Abril 1996

[LifeCycle97] CORBA Services: Lifecycle Service

ftp://ftp.omg.org/pub/docs/formal/97-02-11

[Lingnau95] A. Lingnau and O. Drobnik: "An Infrastructure for Mobile Agents: Requierements and Arquitecture", Proc. 13th DIS Workshop, Orlando, Florida, Setembro 1995.

[Lugmayr99] Wolfgang Lugmayr: "Gypsy: A Component-oriented Mobile Agent System ", tese de doutorado. http://www.luxnet.at/lux/docs/gypsy-diss.ps.gz, 1999.

[MASIF98] Mobile Agent System Interoperability Facilities Specification, Joint Submission, OMG RTF doc. orbos/98-03-09, Março, 1998.

[Mendes96] Manuel Mendes et al, "Agent skills and their roles in mobile computing and personal communications", Mobile Communications - Technology, tools, applications, authentication and security. pp. 181-204, Chapman&Hall, 1996 (IFIP World Conference on Mobile Communications Canberra, Australia, Setembro 2-6, 1996).

[Méndez97] Programa em Java para comprimir/descomprimir arquivos.

http://www.umag.cl/~carias/java/MainZip1.java

[Messaging98] Asynchronous Messaging Service.

ftp://ftp.omg.org/pub/docs/orbos/98-05-05

[Milojicic00] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou: "Process Migration", ACM Computing Surveys, Junho 2000.

[Milojicic98] Dejan S. Milojicic, G. Agha, D. Chauhan, S. Guday, and N. Jamal: "Composing Agent-Based Applications and Systems", Distributed Systems Engineering, IEE, 5 (1998), 1-14.

[Milojicic98a] Dejan S. Milojicic, William LaForge, and Deepika Chauhan: "Mobile Objects and Agents (MOA)", proc. of the Fourth USENIX Conference on Object-Oriented Technologies and Systems (COOTS '98), Abril 27-30, 1998, Santa Fe, New Mexico. Distributed Systems Engineering, IEE, 5 (1998), 1-14

[MOF97] Meta Object Facility, OMG doc. ad/97-10-03

http://ftp.omg.org/pub/docs/ad/97-10-03.pdf

[Naming97] "CORBA Services: Naming Service"

ftp://ftp.omg.org/pub/docs/formal/97-02-08

[Nelson97] Mark R. Nelson: "Java and the Zip File Format", Dr. Dobb's Journal, Dezembro 1997.

[Notification98] "Notification Service", ftp://ftp.omg.org/pub/docs/telecom/98-11-01

[Nwana96] Hyacinth S. Nwana: "Software Agents: An Overview", http://www.cs.umbc.edu/agents/introduction/ao

[Oberon99] "Oberon System 3 White Paper", http://www.oberon.ethz.ch/system3/white.html

[Papaioannou98] T. Papaioannou and J. Edwards: "Using Mobile Agent Technology to improve the alignment between the Sales Order Process in a Virtual Enterprise and its IT Systems" International Journal of Robotics and Autonomous Systems, 1998. http://luckyspc.lboro.uk/Docs/Papers/ljras.pdf

[Papaioannou99] Todd Papaioannou: "Móbile Agents: Are they useful for establishing a virtual presence in space", AAAI 1999 Spring Synposium Series. http://luckyspc.lboro.uk/Docs/Papers/aaai-ss99.pdf

[Peine97] Holger Peine and Torsten Stolpmann: "The architecture of the Ara Platform for Mobile Agents", Proceedings of the First Int. Workshop on Mobile Agents, MA'97, Berlim, Alemanha, LNCS Nr. 1219, Springer Verlag 1997

[Petrie96] Charles J. Petrie: "Agent-Based Enginnering, the Web, and Intelligence", IEEE Expert, Dezembro 1996

[Pfister98] Cuno Pfister: "Brief comparison of Component Pascal and Java", Oberon microsystems, http://www.obsoft.net

[POA99] Capítulo 11, "CORBA Specification", v2.3, OMG, Outubro,1999. http://www.omg.org/technology/documents/formal/corba_2.htm

[Property97] CORBA Services: Property Service

ftp://ftp.omg.org/pub/docs/formal/97-02-18

[PSS99] Persistent State Service, OMG doc. orbos/99-03-02

[Rational98] Rational Software Corporation. http://www.rational.com/index.jsp

[RMI99] Remote Method Invocation, v1.2, Sun Microsystems Inc.

http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html

[Rubin96] Ryan Rubin, and Valentin Kissimov: "Object Migration in the Distributed Computing Environment", II Workshop on Mobility and ReplicationLinz - Austria, 8 Julho 1996. http://www.diku.dk/distlab/workshops/wmr96/pgm.html

[Schmidt97] Douglas C. Schmidt, Aniruddha S. Gokhale, Timothy H. Harrison, and

Guru Parulkar: "A High-Performance End System Architecture for Real-Time CORBA", IEEE Communications Magazine, Fevereiro 1997

[Strasser96] Markus Strasser, Joachim Baumann, and Fritz Hohl: "Mole – A Java Based Mobile Agent System", IPVR (Institute for Parallel and Distributed Computer Systems) University of Stuttgart, 23 Outubro 1996

[Strasser97] Markus Strasser and Markus Schwehm: "A Performance Model for Mobile Agent Systems", H. R. Arabnia (Ed.): `Int. Conf Parallel and Distributed Processing Techniques and Applications (PDPTA'97)', CSREA 1997 Volume II, pp. 1132-1140, .

[Strasser98] Markus Strasser, Kurt Rothermel, and Christian Maihöfer: "Providing Reliable Agents for Electronic Commerce", Trends in Distributed Systems for Electronic Commerce 1998: 241-253. IFIP/GI Working Conference, TREC'98, Hamburg, Germany, 3-5Junho 1998, Proceedings. Lecture Notes in Computer Science, Vol. 1402, Springer, 1998, ISBN 3-540-64564-0

[Susilo98] Gatot Susilo, Andrzej Bieszczad, and Bernard Pagurek: "Infrastructure for Advanced Network Management based on Mobile Code", Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, 15-20 Fevereiro 1998.

[Tacoma99] Projeto Tacoma, http://www.tacoma.cs.uit.no/overview.html

[Twhite99] Tony White e Bernard Pagurek, "Emergent Behavior and Mobile Agents", 1999. http://www.sce.carleton.ca/researchers/tony/index.html

[URI94] Universal Resource Identifiers in WWW (URI), Request For Comment 1630, T. Berners-Lee, CERN, Junho1994. http://www.bilkent.edu.tr/pub/WWW/Addressing/rfc1630.txt

[URL94] Universal Resource Locators (URL), Request For Comment 1738, T. Berners-Lee, L. Masinter, M. McCahill, Dezembro 1994. http://www.w3.org/Addressing/rfc1738.txt

[Venners99] Bill Venners."The Jini Vision". Agosto 1999. http://www.javaworld.com/javaworld/jw-08-1999/jw-08-jiniology_p.html

[Villinger96] K. Villinger, and C. Burger: "Generic mobile agents for electronic markets". http://inforge.unil.ch/isdss97/papers/73.htm.

[Vinoski98] Vinoski, Steve: "New Features for CORBA 3.0", Communications of the ACM, Vol. 41, N°10, Outubro 1998.

[Vitek97] J. Vitek, and C. Tschudin: ""Mobile Object Systems, Towards the Programmable Internet", Lecture Notes in Computer Science 1222, Springer-Verlag, 1997.

[Waldo97] J. Waldo et al: "A Note on Distributed Computing", Lecture Notes in

Computer Science 1222, Springer-Verlag, 1997.

[White94] J. E. White: "Telescript technology: The foundation for the Electronic Marketplace", White paper, Technical report, General Magic, Inc.1994

[White98] Jim White: "Mobile Agents White Paper", http://www-iiuf.unifr.ch/~chantem/white_whitepaper/ whitepaper.html

[Wiseman99] Norman Wiseman: "Internet2: Briefing Paper", Abril 1999.

http://www.jisc.ac.uk/pub99/internet2.html

[Xml97] Extensible Markup Language: Part 1. Syntax, W3C Working Draft 30-Junho 1997. http://www.w3.org/TR/1998/REC-xml-19980210

Artigos publicados durante o desenvolvimento deste trabalho

Sven Krause. F. de Assis Silva, , T. Magedanz, R. Popescu-Zeletin, O. Falsarella e **Carlos Arias Méndez**: "MAGNA - A DPE-Based Platform for Mobile Agents in Electronic Service Market", ISADS'97, Abril 1997, Berlim, Alemanha.

Carlos Arias Méndez e Manuel Mendes: "Agent Migration Issues in CORBA Platforms", ISADS'99, Março 1999, Tokio, Japão.

Carlos Arias Méndez e Manuel Mendes: "Agentes Móviles: Un Servicio de Transporte de Agentes Basado en CORBA", III Workshop Chileno en Sistemas Distribuidos, Novembro 1999, Talca, Chile

Lista de acrônimos

API	Application Programming Interface
ATP	Agent Transfer Protocol
C/S	Cliente/Servidor
CCD	CORBA Component Descriptor
CCD	CORBA Component Descriptor
CDR	Common Data Representation
CIDL	Component IDL
CoABS	Control of Agent Based Systems
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSD	CORBA Software Descriptor
DCE	Distributed Computing Environment
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
DTD	Document Type Declaration
FTP	File Transfer Protocol
GIOP	General Inter-ORB Protocol
HTTP	Hyper Text Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
JAR	JAva aRchive
JDBC	Java Database Connectivity
JDK	Java Design Kit
JNDI	Java Naming and Directory Interface
JTA	Java Transaction API
LSB	Least Significant Byte
MASIF	Mobile Agent System Interoperability Facilities
MOF	Meta Object Facility

MSB	Most Significant Byte
OMG	Object Management Group
ORB	Object Request Broker
PC	Personal Computer
POA	Portable Object Adapter
RMI	Remote Method Invocation
RPC	Remote Procedure Call
S.O.	Sistema Operacional
SCA	Serviço de Comunicação de Agentes
SCSL	Sun Community Source Licence
SE	Serviço de Eventos
SEA	Serviço de Execução de Agentes
SGA	Serviço de Gerenciamento de Agentes
SMTP	Simple Mail Transfer Protocol
SN	Serviço de Nomes
SP	Serviço de Propriedades
SRA	Serviço de Registro de Agentes
SSA	Serviço de Segurança de Agentes
SSL	Secure Socket Layer
STA	Serviço de Transporte de Agentes
TCP/IP	Transport Control Protocol/Internet Protocol
UML	Unify Modelling Language
URI	Uniform Resource Identifiers
URL	Uniform Resource Locators
XML	eXtensible Markup Language