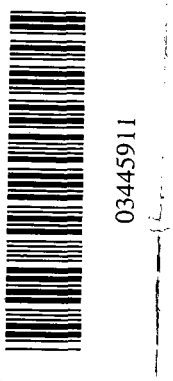
UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Eduardo Augusto Oliveira Lobo

Avaliando a Interoperabilidade de Plataformas de Agentes Móveis Através da Padronização OMG MASIF

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação



Prof. João Bosco Mangueira Sobral, Dr.

Florianópolis, Agosto/2001

Somente com determinação e fé poderemos alcançar
nossos mais distantes objetivos.

Ao meu irmão Célio Henrique e à Anna Luiza pelo grande apoio e incentivo para a realização deste trabalho.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Célio e Heloisa, pelo apoio e o carinho.

Ao meu orientador Prof. João Bosco pelos ensinamentos e sugestões durante a realização deste trabalho.

Aos colegas Edmilson e Marcos Rezende pela atenção dada, nos momentos de exposição dos planos e propósitos.

RESUMO

Este trabalho, além de uma revisão da literatura sobre sistemas de agentes e de uma descrição de cinco plataformas de agentes móveis - Aglets, Voyager, Concordia, Grasshopper e MAP -, contém uma análise crítica sobre a especificação do MASIF (*Mobile Agent System Interoperability Facility*). Um padrão para agentes móveis adotado pelo OMG (*Object Management Group*), que despertou interesse do meio acadêmico e empresarial em razão da especificação que faz com vistas à interoperabilidade de diferentes sistemas de agentes. Para o que, propõe a utilização de duas interfaces: MAFAgentSystem e MAFFinder, a primeira que define padrões para gerenciamento e transferência de agentes, e a segunda, para sistema de nomeação e localização de agentes.

Os testes com as referidas plataformas, que apresentavam indícios de compatibilidade com a especificação MASIF, mostraram que não foi possível o envio de agentes de uma região à outra, mesmo entre diferentes plataformas compatíveis com a especificação MASIF. Entre as conclusões da pesquisa, algumas soluções são indicadas para permitir troca de agentes entre diferentes regiões de plataformas diversas, visando à superação da incompatibilidade identificada.

ABSTRACT

This present work has a throughout review on the literature of Agents Systems, a description of five mobile agent platforms (Aglets, Voyager, Concordia, Grasshopper and MAP) and a critical analysis on the MASIF (Mobile Agent System Interoperability Facility) specification. A standard for mobile agents adopted by the OMG (Object Management Group) that has attracted both the industry and academia. It specifies the interoperability of different mobile agent systems through the utilization of two interfaces: MAFAgentSystem and MAFFinder. The first interface defines standards for agents' management and transfer and the second one for agents' naming and tracking.

Tests made with the referred platforms have shown not to be possible to send agents from one region to another, even when diferent platforms were MASIF compliants. In order to overcome this detected incompatibility, a few solutions were indicated.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ANEXOS

LISTA DE SIGLAS

ANSI	American National Standards Institute
AIS	Adaptative Intelligent Systems
ASCII	American Standard Code for Information Interchange
ATM	Assynchronous Transfer Mode
DARPA	Defense Advanced Research Projects Agency
FIPA	Foundation for Intelligent Physical Agents
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IA	Inteligência Artificial
IP	*Internet Protocol*
ISO	International Organization for Standardization
KQML	Knowledge Query and Manipulation Language
LAN	Local Area Network
MAN	Metropolitan Area Network
MIME	*Multi Purpose Internet Extensions*
MIT	Massachussets Institute of Technology
OMG	Object Management Group
TCP/IP	Transmission Control Protocol/*Internet* Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network

INTRODUÇÃO

O crescimento de redes heterogêneas no mercado tem requerido aumento de confiabilidade e de qualidade de serviço das mesmas, que vem impulsionando o desenvolvimento de novos agentes de *software*. Ao invés de uma grande e centralizada aplicação que abrange toda a inteligência do sistema, um número de sistemas de tamanho relativamente pequeno, chamados agentes - fixos ou móveis, são desenvolvidos para facilitar o acesso a informações espalhadas em diferentes sistemas.

Por meio de agentes móveis pode-se executar tarefas de modo mais econômico, em termos de consumo de recursos da rede ou CPU. Por exemplo: quando uma determinada ação é desejada, um agente - denominado *deglet* -, é enviado, executado localmente e extinto, após cumprir sua tarefa.

A utilização de agentes móveis pode ter vantagens sobre outras implementações de agentes, o que não implica que outras tecnologias não possam ser usadas. Praticamente toda tarefa que pode ser executada com agentes móveis pode ser feita com objetos estacionários, todavia, soluções tradicionais podem ser menos eficientes e mais difíceis de desenvolver.

Os agentes móveis têm aberto um universo de novas possibilidades nas redes de computadores, inclusive na *Internet*, com benefícios quanto a: confiabilidade; qualidade; eficiência; economia; redução no tráfego da rede; interação autônoma assíncrona; robustez e tolerância a falhas; suporte a ambientes heterogêneos; extensibilidade de serviços *on-line*; paradigma de desenvolvimento conveniente e fácil atualização de programas.

Entretanto, existem dificuldades, em virtude da diversidade de implementações de diferentes sistemas de agentes móveis por diversos fabricantes. Assim, como em outros setores da computação, a aceitação e ampla utilização de agentes móveis irão depender fortemente do estabelecimento de padrões.

Para ajudar a solucionar este problema o OMG (*Object Management Group*) especificou o padrão MASIF (*Mobile Agent System Interoperability Facility*), para garantir a interoperabilidade de diferentes sistemas de agentes móveis. Em sua curta história, o OMG MASIF (também denominado apenas MASIF), despertou grande interesse da academia e da indústria, por possibilitar a interoperabilidade de diferentes sistemas de agentes. Nele são especificadas duas interfaces: *MAFAgentSystem* e *MAFFinder*. A primeira refere-se à definição de padrões para o gerenciamento e a transferência de agentes, e a segunda define a padronização para um sistema de nomeação e localização de agentes.

Este projeto teve por objetivo analisar criticamente a especificação OMG MASIF, através da utilização prática de plataformas compatíveis com a especificação, visando determinar se a especificação é uma condição suficiente para a garantia da interoperabilidade das plataformas.

A metodologia da pesquisa incluiu, além do aprofundamento teórico em material bibliográfico disponível, também, em bibliotecas eletrônicas, trabalho prático para a seleção e verificação da adequação das plataformas de agentes móveis à especificação MASIF, visando possibilitar uma análise prática da especificação.

No desenvolvimento do trabalho, um capítulo apresenta o tema relacionado a agentes e sistemas de agentes - com rápido histórico, conceituação, tipologia e funções.

No segundo capítulo foram discutidos as bases, os tipos e a complexidade da interoperabilidade do modelo OMG MASIF, bem como a descrição de sua IDL (Interface Definition Language) e suas interfaces.

No terceiro capítulo foram analisadas plataformas de agentes móveis, com seus elementos básicos, operações fundamentais, pacotes, formas de transmissão e aspectos de segurança, sempre com vistas à percepção de suas compatibilidades com o padrão OMG MASIF.

No seguinte, a interoperabilidade das plataformas foram testadas e os resultados obtidos, analisados conforme referencial teórico construído nas etapas anteriores.

Nas Conclusões foram elaboradas considerações gerais sobre o trabalho, com posicionamento pessoal sobre a especificação MASIF. Momento em que, também, tentou-se responder às seguintes questões:

Quais benefícios a interoperabilidade nos moldes OMG MASIF pode trazer para os usuários de agentes móveis ?

Em que, a falta da interoperabilidade reflete nos sistemas de agentes móveis ?

Quais características técnicas precisam estar presentes nas plataformas para que estejam em conformidade com o padrão OMG MASIF ?

As plataformas que alegam compatibilidade com o padrão OMG MASIF interoperam em condições plenas ?

1 - AGENTES / SISTEMAS DE AGENTES

1.1 - Evolução dos Agentes

A concepção de agente de *software* vem, desde os anos 1970, com o modelo de objeto proposto por Carl Hewitt denominado "ator". O qual possuiria: autonomia, interatividade e execução concorrente.

> *É um agente computacional que possui um endereço e um comportamento. Atores podem se comunicar e executar suas tarefas concorrentemente* (Hewitt Apud Nwana, 1996:2).

Segundo Nwana (1996) duas linhas de pesquisa foram abertas. A primeira em 1977 até os dias de hoje e a segunda em 1990, também, até os dias de hoje.

A primeira concentrou-se em assuntos como interação e comunicação entre agentes, a distribuição de tarefas, coordenação e cooperação além da resolução de conflitos. Os objetivos eram especificar, analisar, projetar e integrar sistemas envolvendo múltiplos agentes colaborativos. Caracterizou-se, também, pela pesquisa e desenvolvimento de aspectos de teoria, arquitetura e linguagem.

Todavia, desde 1990, uma segunda linha de pesquisa e desenvolvimento de programas agentes se iniciou ampliando a gama de tipos de agentes. Reconhece-se que numerosos tipos de programas são chamados de agentes e nem todos de fato o são.

Um grande número de universidades e centros de pesquisa dedicam considerável esforço no sentido de desenvolver programas agentes, entre as quais pode-se citar: Carnegie Mellon University/EUA; Massachussets Institute os Technology (MIT)/EUA; The University of London; Carleton University/Canadá; General Magic Inc/USA; Alcatel; Apple; Lucent Technologies; DEC; Hewlett Packard; IBM/Lotus; Microsoft; Oracle; Reuters.

A quantidade de aplicações em desenvolvimento é bem ampla, impressiona e vai desde agentes simples até agentes "inteligentes". Por exemplo, o Lotus Notes inclui agentes que permitem aos usuários escreverem seus próprios *scripts* para manuseio de mensagens de correio eletrônico, calendários e estabelecimento de reuniões. Outro tipo de sistema de agentes é o *Sycara's visitor hosting system* da Carnegie Mellon University/EUA. Este sistema utiliza agentes para manuseio de tarefas e de informações que cooperam entre si para montar uma tabela com informações do visitante. Para atingir este objetivo, os agentes acessam fontes de informação *on-line*, visando determinar áreas de interesse, nome e organização à qual pertence

o usuário assim como resolver inconsistências e ambigüidades. Mais informações são depois detalhadas incluindo dados sobre a posição do visitante na organização e projetos em que esteja envolvido.

Para a marcação de uma reunião entre membros da vários departamentos muitas etapas podem ser automatizadas, por exemplo: determinação de pessoas envolvidas no assunto em questão, determinação do horário mais adequado para cada uma delas e estabelecimento do horário da mesma. Primeiramente para determinação das pessoas envolvidas é feita uma consulta a um banco de dados com informações coletadas previamente, tal como quais pessoas em quais departamentos estão envolvidas em que projetos.

Após determinar quem irá participar da reunião, agentes de calendário são enviados aos respectivos envolvidos/departamentos para coleta de informação sobre horários disponíveis. Cada agente coleta a informação sobre a disponibilidade da pessoa e retorna à origem para concentração da informação e estabelecimento do melhor horário. Nesta etapa várias considerações são feitas. Por exemplo: determinadas pessoas preferem não ter reuniões às segundas pela manhã ou às sextas na parte da tarde, outras têm indisponibilidades devido a função ou horário de trabalho.

Depois do retorno dos agentes com as informações de disponibilidade de cada pessoa, o agente ligado à tarefa de marcação da reunião, em acordo com parâmetros estabelecidos, decide o horário para reunião e solicita confirmação ou oferece opções a quem é de direito. Eventuais incompatibilidades são resolvidas utilizando os parâmetros, podendo levar em conta a hierarquia de cada um dos envolvidos. Finalmente os envolvidos são informados sobre o horário estabelecido.

À medida que outras aplicações forem sendo descritas, outros modelos de agentes serão também descritos, tais como atividades de gerenciamento de redes, controle de tráfego aéreo, gerenciamento de processos de fabricação nos mais variados setores da indústria, sistemas de vigilância e de suporte à vida, *data mining*, gerenciamento e recuperação de informações, comércio eletrônico, educação, *personal digital assistants* (PDA), correio eletrônico, bibliotecas digitais, gerenciamento de horário/agenda, entre outros.

O potencial de mercado estimado pela Ovum's/Reino Unido, uma companhia de pesquisa de mercados, em 1994, era de que o volume de mercado para EUA/Europa em 1995 seria de 475 milhões de dólares americanos, com uma projeção estimada de 3,9 bilhões de dólares para o ano 2000 (www.ovum.com/reports).

No MIT , o grupo liderado por Pattie Maes trabalha com agentes que relacionam vendedores e compradores de determinado produto estão também desenvolvendo pesquisas que utilizam teorias de evolução biológica para implementar demonstrações que somente agentes que se enquadrem a contento irão se reproduzir, ao passo que os mais fracos serão eliminados.

Alguns dos agentes aqui descritos são ainda uma visão de futuro, espera-se, não muito distante. Considera-se que os agentes de *software* vieram para ficar e conforme dito por Nwana:

> *... estamos nos movendo mais e mais em direção a era da informação, e qualquer instituição/empresa baseada na informação e no conhecimento que não invista em tecnologia de agentes estará cometendo um hara-kiri.* (Nwana. Knowledge Engineering Review,1996:5)

1.2 - *Software* Agente

É difícil definir o que são agentes. Uma das razões para isso é a ampla e heterogênea gama de pesquisas que se utilizam do termo "agente". Para resolver isso, alguns pesquisadores se utilizam de termos como *softbots* (*software robot*), *taskbots* (*task-based robots*), *userbots*, *robots*, *personal agents*. Para isso utilizam-se de definições tais como: caso ocupem o mundo físico são denominados *robots*, se habitam nas redes de computadores são denominados *softbots*. Os que executam tarefas específicas são muitas vezes chamados de *taskbots*. *Autonomous agents* referem-se a agentes móveis ou *robots* que operam em ambientes dinâmicos.

> *Um agente é um sistema de computador encapsulado que está situado em algum ambiente, e que é capaz de uma ação autônoma e flexível neste mesmo ambiente com o objetivo de alcançar seus objetivos estabelecidos* (Jennings, 1999:2).

Ou como descrito no MASIF (cap. 1, p. 5):

> *Um agente é um programa de computador que age autonomamente em favor de uma pessoa ou organização. Muitos dos agentes são programados em linguagens interpretadas (por exemplo: Tcl e Java) por portabilidade.*

Uma outra razão é que agentes podem executar tarefas complexas como assistentes pessoais, para isso dominando o conhecimento de algum campo específico. Também devido à multiplicidade de tarefas existe uma avalanche de nomes que podem ser atribuídos, tais como:

search agents, *report agents*, *presentation agents*, *navigation agents*, *management agents*, *search and retrieval agents*, *development agents*, *analysis and design agents*, *testing agents*, *packaging agents*, *help agents*. Caso eles executem: buscas, relatórios, apresentações, navegação, gerenciamento, buscas e tratamento da informação, algum tipo de desenvolvimento, análise e projeto, testes ou algum tipo de ajuda.

1.2.1 - Conceituação de agente

Um agente poderia ser conceituado como entidade computacional, programa ou *máquina*, que age em benefício de outros pró-ativamente ou reativamente e exibe capacidade de aprender, cooperar e mover-se. Estas características básicas definem um modelo de agente básico. Alternativamente poderia-se afirmar que agentes podem ser considerados como uma classe que engloba um grande número de tipos de agentes e então partir-se-ia para uma definição de cada um destes tipos.

> *Agentes de software originaram-se de sistema multi-agentes e da inteligência artificial que envolviam processamento paralelo. Da Computação Distribuída herdaram a modularidade, a velocidade (em razão do paralelismo) e confiabilidade (em razão da redundância); da Inteligência Artificial vieram a capacidade de aprendizagem, a facilidade de manutenção, reusabilidade e a independência de plataforma* (Huhns & Singh, 1994, citado por Nwana,1996).

1.2.2 - Tipos de Agentes

Muitos fatores devem ser considerados para classificar corretamente os agentes.

Quanto à sua mobilidade, podem ser classificados como agentes **móveis** ou **estáticos**. Os primeiros possuem habilidade de mover-se por uma rede, ou seja, não esta restrito ao sistema onde inicia sua execução. Os segundos tem sua execução restrita ao sistema onde a execução foi iniciada.

Os agentes podem, também, ser **reativos** ou **deliberativos**. Caso sejam reativos, agem unicamente quando recebem algum estímulo alterando seu estado dentro do ambiente em que se encontram. Quando deliberativos, possuem um modelo simbólico interno e podem se engajar em um planejamento ou negociação junto com outros agentes, com vistas ao cumprimento da tarefa.

Em terceiro lugar, podem ser classificados de acordo com alguns parâmetros ideais ou primários que os agentes devam possuir, tais como: **autonomia, cooperação e aprendizagem**. Autonomia refere-se à capacidade do agente em executar sua tarefa sem ajuda

humana. Os agentes podem possuir estados internos e objetivos próprios, e agem de modo a atingir estes mesmos objetivos em benefício do usuário. A cooperação com outros agentes é de grande importância, pois em uma mesma rede pode-se ter vários agentes em tarefas diferentes ou similares, na qual uma comunicação e troca de informação entre eles pode resultar em um grande benefício ao usuário. Os objetivos de cada um deles podem ser completados de maneira eficiente, para isso é necessária entre eles algum tipo de linguagem de comunicação. A aprendizagem refere-se à capacidade do agente em aprender a interagir com o ambiente. Esta aprendizagem pode ser vista como uma melhoria de performance na execução de determinada tarefa.

De acordo com Nwana, estas três características podem ser utilizadas para definir quatro tipos de agentes (Fig.1), como descritos a seguir. Para cada combinação de duas características é definido um tipo de agente o que leva à definição de três tipos de agentes. E quando possui as três características é definido um quarto tipo.

- Autonomia e aprendizagem - agentes de interface.
- Aprendizagem e cooperação - agentes colaborativos com capacidade de aprendizagem.
- Cooperação e autonomia - agentes colaborativos.
- Autonomia, aprendizagem e cooperação - agentes inteligentes.

A distinção destas características não é definitiva. Um exemplo é que agentes colaborativos têm uma maior ênfase em cooperação e autonomia. O que não quer dizer que não possuam capacidade de aprendizagem. Não se considera agente qualquer outro tipo de programa que esteja fora da interseção de pelo menos duas dessas características.

Pode-se combinar as diferentes classificações estático/móvel, colaborativo/interface e obter-se algo do tipo: agente colaborativo deliberativo estático, agente colaborativo reativo móvel ou ainda agente de interface reativo móvel.

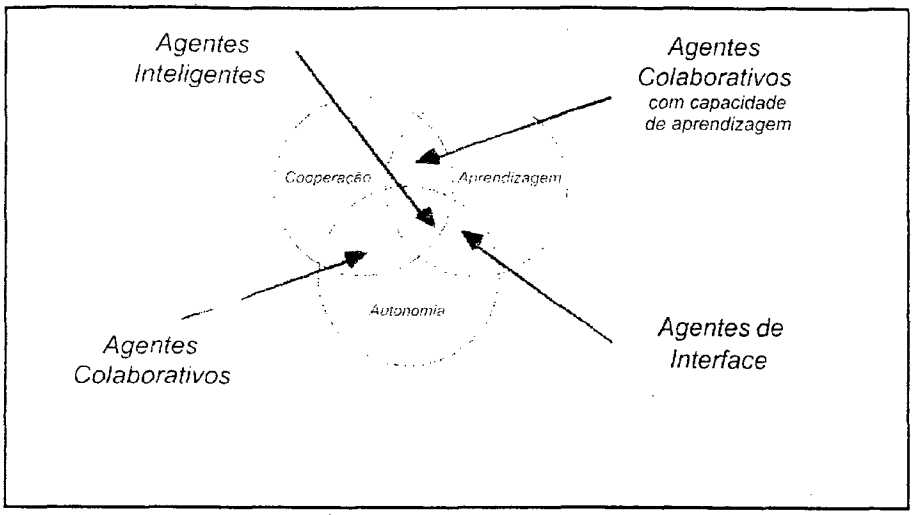


Figura 1 - Visão parcial de uma tipologia de agentes. (Nwana, 1996:6)

Em quarto lugar, os agentes podem ser classificados pela sua **função principal**. Por exemplo: agentes de informação de *Internet*. Esta categoria engloba mecanismos de busca utilizados por renomados *sites* de busca na *Internet* tais como: WebCrawler e Lycos baseados no exterior e Miner baseado no Brasil. Este tipo de agente pode ser ainda do tipo estático ou móvel.

Em quinto lugar, pode-se considerar um tipo adicional de agente, o **híbrido**. Tal agente pode ser assim classificado, caso possua as características de mais de uma categoria em um único agente.

Outras questões podem surgir no que tange à classificação de agentes. Podem ser **benevolentes**, **egoístas**. Podem,também, mentir deliberadamente ou não.

Pode-se afirmar que os agentes não necessitam ser benevolentes uns com os outros. Eles podem competir entre si e podem, inclusive, ter ações antagônicas. Agentes que competem entre si podem pertencer a qualquer um destes tipos. Podem ser, por exemplo, agentes de informação ou de interface que competem com outros agentes do mesmo tipo ou de tipos diferentes.

Levando-se em consideração os tipos de agentes já existentes e os que se espera existirem logo, pode-se elaborar uma lista, a princípio arbitrária, que contém sete tipos de agentes:

- Agentes **colaborativos**
- Agentes de **interface**
- Agentes **móveis**

- Agentes de **informação/*Internet***
- Agentes **reativos**
- Agentes **híbridos**
- Agentes **inteligentes**

Já Wooldridge & Jennings (citado por Nwana, 1996) classificam os agentes de uma maneira mais ampla e simplista. Os agentes são classificados em agentes *gopher*, agentes para prestação de serviços e agentes pró-ativos.

De um modo geral, os programas que não se encaixam em nenhuma destas categorias não devem ser considerados agentes. Sistemas especialistas, por exemplo, não são classificados como agentes, sobretudo pela falta de autonomia entre seus módulos.

Os **agentes colaborativos** tem sua ênfase em autonomia e cooperação com outros agentes. Para isso devem possuir um padrão de negociação que lhes permite entrarem em acordo para que seus objetivos possam ser atingidos.

Em geral, as características destes agentes são: autonomia, habilidades sociais, pronta reação às solicitações e pro-atividade. Por isso eles podem agir racionalmente e autonomamente em um ambiente com múltiplos agentes. Tipicamente muitas das implementações feitas com este tipo de agente não envolvem atividade de aprendizagem complexa mas podem, ou não, ter algum aprendizado limitado a parâmetros ou rotas.

Os objetivos deste tipo de agente podem ser:

- Resolver problemas que podem ser muito grandes para um agente único e centralizado, ou ainda dividir os riscos de se ter um sistema centralizado.
- Permitir a interconexão e interoperabilidade de múltiplos sistemas legados - tais como em: sistemas especialistas ou em sistemas de apoio a decisões.
- Fornecer soluções para problemas distribuídos, como necessário em rede de sensores distribuídos ou em controle de tráfego aéreo.
- Para prover soluções para sistemas *on-line* que se utilizam de bases da dados distribuídas através de uma abordagem com agentes colaborativos distribuídos.
- Fornecer soluções em casos que o conhecimento está distribuído, como no apoio à elaboração de diagnóstico de doenças, através da elaboração de uma lista de possíveis causas.

- Permitir maior: modularidade, devido à redução da complexidade; velocidade, devido ao paralelismo; confiabilidade, devido à redundância; flexibilidade (novas tarefas podem ser elaboradas mais simplesmente) devido à organização modular.
- Na elaboração de modelos para outras áreas. Por exemplo, no estudo de relações humanas ou biológicas.

Um exemplo de aplicação de agentes com estas características é o projeto Pleiades, da Carnegie Mellon University/EUA. Um de seus objetivos é o de investigar métodos para automatizar a negociação entre agentes colaborativos, visando implementar sua robustez, efetividade, escalabilidade e manutenção.

O Pleiades é uma arquitetura que se utiliza de agentes colaborativos distribuídos, organizados em duas camadas abstratas (Fig.2). A primeira camada contém os agentes relacionados a tarefas e a segunda contém agentes relacionados a informações. Esta é a arquitetura utilizada no *Sycara's visitor hosting system.*

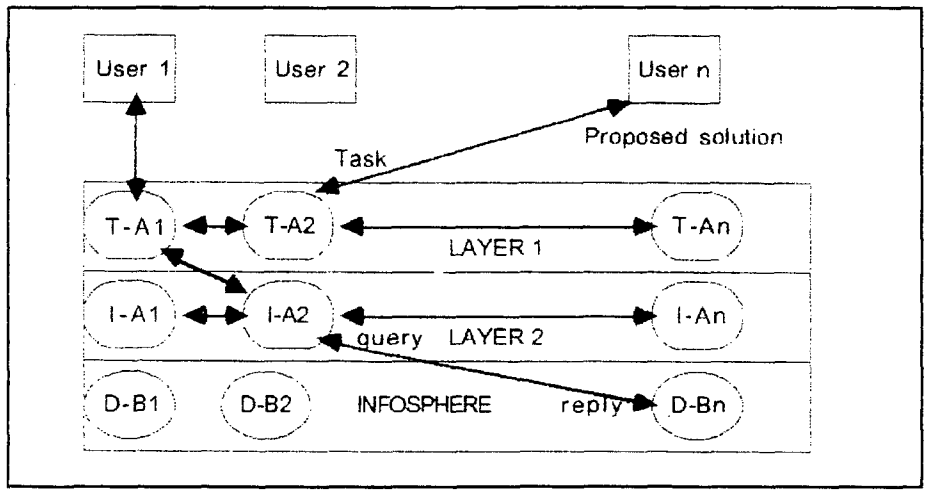


Figura 2 - Sistema de arquitetura distribuída Pleiades (Nwana, 1996:11, adaptado de Sycara, 1995).

Os agentes da primeira camada executam tarefas específicas solicitadas pelo usuário, tais como marcação de reuniões ou encontros, coordenando e programando planos baseados em um contexto. Estes agentes colaboram com outros agentes desta mesma camada com o intuito de resolver conflitos e integrar informação. Cada agente desta camada, incluindo o que atendeu o usuário, solicita informações dos agentes de informação presentes na segunda

camada. Assim como na primeira, os agentes de informação da segunda camada também podem colaborar uns com os outros, de modo a prover a informação solicitada para o agente da camada 1. A fonte de informação usada pelos agentes de informação estão distribuídas em muitos bancos de dados. Finalmente, o agente encarregado da tarefa propõe uma solução para o usuário.

Para completar esta tarefa os agentes devem possuir certas características, conforme descrito a seguir:

Agentes de tarefa (*task-agents*) devem possuir: um modelo do domínio da tarefa a executar; conhecimento de como executar a tarefa; conhecimento de como buscar as informações necessárias; conhecimento de outros agentes de tarefa e de informação que poderá coordenar; protocolos para comunicação com os outros agentes de ambas as camadas; estratégias para resolução de conflitos e agrupamento de informações; podem ter ainda alguma capacidade de aprendizagem.

Agentes de informação (*information-agents*) devem possuir: conhecimento do banco de dados ao qual possui acesso (tamanho, tempo médio de acesso, custos, etc); conhecimento de como acessar este banco de dados; estratégias para resolução de conflitos e agrupamento de informações; protocolos para comunicação e coordenação de outros agentes; inteligência suficiente para formar uma memória *cache* de dados acessados com maior freqüência.

A motivação deste tipo arquitetura é proporcionar acesso, recuperação e seleção de dados distribuídos e permitir apoio a decisões. Para que isso ocorra, Sycara (citado por Nwana, 1996) e outros membros da equipe na CMU (Carnegie Mellon University) formularam a hipótese de que para atingir os objetivos, os agentes seriam capazes de coletar, filtrar e unir informações, além de aprender com estas interações. Para comunicarem entre si, os agentes podem utilizar a linguagem KQML (*Knowledge Query and Manipulation Language*) e correio eletrônico utilizando codificação MIME (*Multi Purpose Internet Mail Extensions*). Para Sycara, a divisão em camadas é claramente modular, o que permitiu a inclusão de módulos de conexão no projeto de outros sistemas.

Individualmente, cada agente consiste de um módulo de planejamento ligado a uma base de crenças e fatos. Possui também um programador local, um módulo de coordenação e um monitor de execução. Deve também poder instanciar tarefas e coordenar sua execução junto a outros agentes monitorando e programando a execução de operações locais.

Apesar das muitas dificuldades na implementação de agentes deste tipo, várias aplicações já passaram por testes bem sucedidos. Um protótipo para gerenciamento de tráfego aéreo, denominado OASIS (*Optimal Aircraft Scheduling using Intelligent Systems*), foi testado com sucesso no aeroporto de Sidney/Austrália. Foram testados mais de cem agentes de aeronaves e mais de dez agentes globais que lidavam com questões como velocidade e direção de ventos, trajetórias e coordenadas. Muitas áreas podem vir a se beneficiar de tais tipo de agentes, em especial áreas que envolvam informações distribuídas e grande relacionamento entre os agentes, o que ocorre em muitos setores que envolvem atividades de gerenciamento com informações oriundas de fontes diferentes.

Os **agentes de interface** têm sua ênfase em autonomia e aprendizagem para a execução de suas tarefas de apoio aos usuários. Note-se que, enquanto os agentes colaborativos preocupam-se muito com o apoio a outros agentes, os agentes de interface preocupam-se com o apoio ao usuário.

Basicamente, o agente monitora o que o usuário está fazendo para, então, assumir o que o usuário está executando, ou então sugerir a maneira melhor de fazê-lo.

O agente pode aprender a fazer algo de várias maneiras: observando e imitando as ações do usuário; através de um retorno, positivo ou negativo, dado pelo usuário; por instrução explícita do usuário; questionando outros agentes que podem estar na própria máquina do usuário ou em qualquer ponto da rede, inclusive *Internet*. Com isso, o agente pode ter inicialmente uma base de conhecimento mínimo, adquirindo mais conhecimento à medida que necessite.

A cooperação entre agentes deste tipo se dá somente pela busca de sugestões ou consultas, não havendo nenhum mecanismo de negociação como ocorrem com os agentes colaborativos.

O objetivo é que o usuário, novato ou não, possa receber ajuda e/ou possa ser assistido pelo agente. Deste modo, o agente pode assumir a tarefa ou pelo menos a parte mais entediante, liberando o usuário para desenvolver seu trabalho. O agente faz então o importante papel de interface entre o usuário e o aplicativo em utilização, permitindo ao usuário dedicar-se mais ao desenvolvimento de suas tarefas e menos com à interação com os programas.

Os benefícios obtidos pelo usuário na utilização de tal tipo de agente pode ser visto de três modos: menos trabalho para o usuário; adaptabilidade do agente com os hábitos e atitudes do usuário; conhecimentos adquiridos por um agente podem ser partilhados com outros através de uma base de conhecimento ao qual todos tenham acesso.

A seguir são apresentados seis casos de aplicações com este tipo de agente (citado por Nwana, 1996):

- Agente de calendário (Kozierok & Maes, 1993) - Assiste o usuário em programação de tarefas e encontros que envolvam aceitação, rejeição, agendamento, negociação e re-agendamento.
- *Letizia* (Liebermann, 1995-2000) - É um agente de interface que auxilia o usuário durante uma pesquisa pela *Internet*. Utilizando uma exploração autônoma à frente do usuário baseado em seus interesses em pesquisas anteriores e fazendo inferências. Estas inferências são continuamente atualizadas em função do interesse que o usuário venha a ter por determinadas páginas ou assuntos durante a busca. O grande feito do *Letizia* é a redução do tempo de busca, pois no método tradicional o usuário fica aguardando o levantamento de resultados durante a busca, enquanto o mecanismo de busca fica aguardando enquanto o usuário consulta a lista retornada pelo referido mecanismo.
- *The Remembrance Agent* (Rhodes & Starner, 1996-2000) - é um sistema de agente pró-ativo de recuperação de informações que trabalha em conjunto com editores de correio eletrônico. Após algumas mensagens escritas pelo usuário, o agente é capaz de recomendar informações úteis através de buscas baseadas em palavras. Através da seleção das cinco mensagens mais relevantes e relacionadas à mensagem que está sendo escrita, o mesmo recomenda a inclusão de *links* e/ou anexos que podem estar sendo esquecidos de serem incluídos. É sobretudo um ajudante à memória de quem está escrevendo as mensagens pois recomenda a inclusão inclusive de artigos que foram citados em mensagens anteriores e que poderiam ser citados também nesta em elaboração.
- *Yenta/Yente-lite matchmaking agent prototype* (Foner, 1996-2000) - o objetivo deste *software* agente é agir como intermediário promovendo o contato entre pessoas, tais como na promoção de encontro de compradores e vendedores de um mesmo produto, ou ainda para colocar em contato pessoas com interesses semelhantes. Cada usuário nesta comunidade possui um agente que carrega consigo algumas informações e referências sobre o usuário. Isto é feito através de uma arquitetura multi-agente, descentralizada e

peer-to-peer. Para isso, tais agentes devem encontrar uns aos outros e executar uma união de informações úteis sem saber onde estão os demais agentes (deste tipo) na rede.

- *Kasbah* (Chavez & Maes, 1996) - é um agente que lê anúncios na *Internet* que incorporam agentes. Atuando como um mercado ou bolsa de negócios, ele pode comprar ou vender de acordo com parâmetros passados pelo usuário. Os parâmetros podem ser algo como: em uma compra o valor máximo a ser pago; em uma venda o valor mínimo; quantidade máxima e mínima de um determinado item e limites de desembolso. A intenção é de que o mesmo possa executar as funções de um corretor de modo mais automático. Para isso deve dispor de técnicas tais como: verificar todos os anúncios sobre um dado item, ao invés de comprar do primeiro que encontrar; dispor de mecanismo de negociação visando evitar a venda sempre pelo valor mínimo e a compra sempre pelo valor máximo estabelecido.
- *Ringo/HOMR*[1] *system* (Shardanand & Maes, 1995) - é um sistema personalizado de busca de músicas e artistas que se utiliza de agentes de interface. Quando o usuário busca determinado artista ou música, o agente fornece uma lista de tópicos relacionados. Por exemplo: ao procurar por determinado artista, o agente retorna uma lista de músicas, álbuns deste artista e *shows* realizados, assim como outros artistas relacionados.

Agentes de interface possuem grande possibilidade de aplicação na indústria do entretenimento, em aplicações noticiosas e em apoio a usuários novos e experientes na utilização de programas. Grande sucesso já foi experimentado por várias aplicações, como por exemplo o *Ringo/HOMR* que foi usado por mais de 2000 pessoas durante sua fase de testes (Shardanand & Maes, 1995).

Alguns desafios são impostos para a utilização deste tipo de agente, tais como: que o conhecimento adquirido pelos agentes venham realmente a facilitar o trabalho do usuário, reduzindo sua carga de trabalho; que os usuários realmente queiram utilizá-los; possibilitar uma melhoria contínua na competência dos agentes, tal que os usuários venham a confiar no trabalho realizado por eles; estender a utilização deste tipo de agente a aplicações tão vastas quanto inovativas, como feitas pelo *HOMR*.

---

[1] *Helpful Online Music Recommendation Service*.

Os **agentes móveis** podem ser definidos como programas capazes de se deslocar em redes de computadores (LAN, MAN, WAN), inclusive *Internet*, interagindo com outros sistemas, coletando informações ou executando outras tarefas, podendo retornar à origem ou não a critério do proprietário. Podem mover-se entre localizações diferentes, ou seja, eles não se restringem ao sistema onde sua execução foi iniciada. Têm a habilidade de transportar-se de um sistema em uma rede, para outro.

Em complemento ao modelo básico, todo *software* agente define um modelo de ciclo de vida, de segurança e de comunicação, e para agentes móveis um modelo de navegação.

O quadro 1, a seguir, adaptado da tabela do *Perpetuum Mobile Procura* da *Carleton University*, lista alguns tipos de agentes:

Tipo de Agente	Descrição
Extlet	Extensão de código que expande a parte adicional que recebe via rede.
Netlet	Código móvel que migra entre os elementos da rede executando suas funções.
Deglet	Código móvel que executa uma atividade determinada em um local determinado e se extingue após cumprir sua função.
Servlet	Expansão de código que amplia as capacidades de um servidor remoto estendendo seu protocolo de interface.
Applet	Código móvel que representa uma aplicação enviada via rede.
Piglet	Código móvel que foi interceptado e maliciosamente alterado.

Quadro 1 - Descrição de alguns tipos de agentes móveis.

Para o modelo de ciclo de vida, há a necessidade de se prover serviços de criação, inicialização, pausa, parada ou destruição de agentes.

Para o modelo computacional, há a necessidade de se controlar as capacidades de cada agente no que se refere a manipulação de dados e limites de atuação.

Para o modelo de segurança, há a necessidade de descrever os modos como o agente pode acessar os recursos da rede, assim como acessar outros agentes na rede.

Para o modelo de comunicação, define a comunicação entre agentes e entre agentes e outras entidades.

Para o modelo de navegação, define o modo de transporte dos agentes entre as diferentes localidades.

Para fazer uso de agentes móveis, um sistema deve possuir uma plataforma que possibilite esta mobilidade de agentes originados de diferentes sistemas de agentes.

Para ser um agente móvel, o programas deve possuir não somente a mobilidade, uma condição necessária mas não suficiente, mas também a autonomia e a capacidade de cooperação. Esta cooperação é feita através da divulgação da localização de alguns dos dados de posse para outros agentes. Isto permite que os mesmos acessem informações, ou parte delas, sem expor totalmente o agente que as divulga.

Uma das grandes vantagens dos agentes móveis é a redução do tráfego na rede e dos recursos computacionais exigidos, mas muitos outros podem ser citados: simples coordenação; computação assíncrona; flexível e distribuída arquitetura de computação; redução da complexidade, etc.

A redução no tráfego da rede é conseguida por meio do envio dos agentes e dos dados selecionados pela rede. É o que ocorre na solicitação de uma imagem: o agente é enviado até a fonte dos dados, efetua a seleção localmente e envia a imagem selecionada a quem a solicitou. No método tradicional ao ser efetuada tal busca, muitas imagens seriam enviadas pela rede até quem as solicitou para que a seleção pudesse ser feita, manualmente ou por agentes fixos. Isso diminui enormemente o volume de dados que trafegam pela rede, o que é especialmente importante para quem paga pelo volume de dados transmitidos, pela largura de banda disponível ou utiliza conexões discadas, sobretudo quando se possui uma quantidade limitada de recursos computacionais, principalmente memória e capacidade de processamento. A máquina de destino também tem seus recursos poupados, uma vez que o agente representa um programa relativamente pequeno e os dados a enviar são somente os dados selecionados o que permite uma liberação mais rápida dos recursos do servidor.

> *A abordagem de agentes móveis continua a intrigar a mostra sinais de oferecimento de importantes vantagens qualitativas para serviços em redes. Assumindo que soluções aos problemas de segurança sejam resolvidos - e vários esforços estão em andamento - os sinais são suficientemente positivos para mostrar que não podemos escapar da possibilidade de que agentes móveis serão um bem sucedido método de interação entre cliente-servidor em redes de computador.* (Harrison, Chess and Kershenbaum, 1995:19).

O trabalho com agentes móveis tem uma coordenação mais simples, uma vez que pode enviar vários agentes a diferentes destinos e localmente ter somente que agrupar os dados recebidos. A computação pode ser assíncrona, uma vez que o agente pode ser despachado para realizar sua tarefa enquanto seu emissor realiza outra tarefa ou permanece desconectado. O

agente pode, por exemplo, retornar a resposta via mensagem no correio eletrônico. Provê uma arquitetura distribuída e flexível para implementação de numerosos serviços. Novos serviços podem ser criados e extintos na medida que sejam necessários ou desnecessários sem prejuízo do ambiente de implementação, pois novos agentes podem coexistir com outros presentes no mesmo ambiente. Oferece uma alternativa de arquitetura distribuída flexível, diferente das demais que usam abordagem estática.

A mobilidade do agentes é efetuada de diferentes modos dependendo da implementação existente. De um modo geral, o agente é uma entidade autônoma e depende de uma plataforma para funcionar. Para mover-se de um lugar para outro, duas são as abordagens: a) o agente ao encontrar uma instrução para mover-se, suspende sua execução, é compactado, copiado para o endereço de destino, descompactado, reativado sua execução; e, b) em outra abordagem, ao ter que se mover, o agente aciona recursos da plataforma presentes na origem e no destino a partir do qual é duplicado após teste de confirmação, quando a cópia original é apagada e a cópia do destino tem sua execução ativada.

Muitos são os tipos de plataformas para agentes móveis disponíveis. Neste trabalho, maior atenção será dada às plataformas baseadas em Java, uma linguagem de programação baseada em C++, da Sun Microsystems.

Agentes móveis são escritos normalmente em linguagens orientadas a objeto, embora seja possível escrevê-las em outras linguagens. São conhecidos sistemas de agentes móveis escritos em C/C++, Java, Xlisp, Tcl e suas variantes Agent-Tcl, Safe Tcl.

A arquitetura Telescript ilustrada na figura 3, se utiliza de *script* Tcl, elaborado pela General Magic, na qual o Telescript opera sobre uma API (*application programmer interface*) que opera sobre um sistemas operacional. As aplicações Telescript consistem de agentes operando em um "mundo" de lugares(locais), mecanismos e regiões. Um local é uma instanciação de alguma classe em um mecanismo cuja definição herda operações que podem ser chamadas por este mesmo local. O mecanismo é em si um interpretador multi-tarefa que pode executar e alternar entre múltiplas tarefas. Um local pode ser ainda um processo que pode conter outros locais.

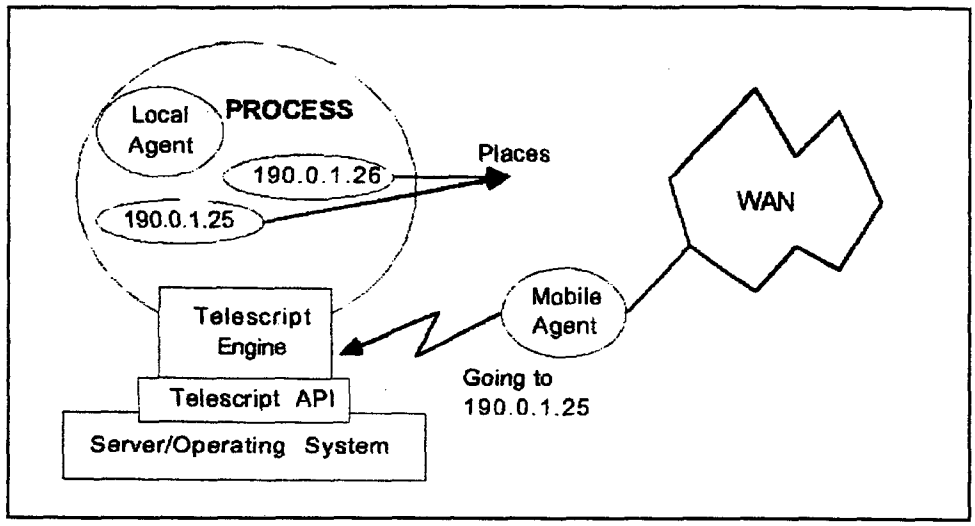


Figura 3 - Vista parcial da arquitetura Telescript (Nwana, 1996:20, adaptado de Wayner, 1995).

A utilização de agentes móveis envolve várias questões:

- Transporte - Como um agente se move de um lado para outro. Como ele se prepara para ser transportado e se move.
- Autenticação - Como garantir que o agente é realmente quem ele diz que é. E que representa quem ele diz que representa.
- Privacidade - Como garantir a privacidade com a utilização de agentes. Como garantir que alguém não autorizado não leia seu conteúdo. E que o agente não seja destruído e substituído por um impostor.
- Segurança - Como protegê-los de vírus. E como prevenir para que não entrem em *loop* eterno.
- Custos - Como o agente irá pagar pelos serviços utilizados. Como assegurar que os agentes não se descontrolem e contraiam enormes valores a pagar.
- Performance - Qual seria o efeito de se ter, centenas, milhares ou milhões de agentes em uma WAN.
- Interoperabilidade - Como prover listas de mecanismos de busca e serviços específicos. Como executar um agente escrito em uma linguagem em uma plataforma escrita em outra. Como publicar ou assinar serviços. Como suportar *broadcasting*.

Algumas das questões anteriores já têm algum tipo de solução por alguma das implementações já feitas, embora outras soluções sejam sempre bem-vindas. Por exemplo, o

Telescript Development Enviroment, da General Magic/EUA, utiliza codificação ASCII, de scripts Tcl-safe e MIME para transporte de mensagens, chave privada digital para autenticação, além de restrições impostas pelo ambiente, tais como não permissão de acesso direto à memória.

Os vários tipos de agentes móveis existentes e as soluções adotadas, serão descritos em detalhe mais adiante.

A principal finalidade dos **Agentes de Informação/*Internet*** é lidar com o crescente volume de informações a ser manipulado, selecionado e agrupado, oriundo, muitas vezes, de fontes diversas.

Alguma discussão tem havido no meio científico quanto à diferenciação de agentes colaborativos e de agentes de informação. Com a crescente utilização da *Internet* como meio de informação *on-line,* alguma confusão tem havido. Os colaborativos são definidos pelo que são enquanto os agentes de informação/*Internet*, são definidos pelo que fazem. Por exemplo: O laboratório Media Lab. no MIT/EUA desenvolveu agentes autônomos e que possuem capacidade de aprendizado, contudo como foram desenvolvidos em termos de *Internet*, eles são considerados agentes de informação.

A frase: "Estamos nos afogando em informação mas com fome de conhecimento" de John Naisbitt, da Megatrends explica a motivação dos agentes de informação/*Internet*. Certamente este tipo de agente pode ajudar, mas não resolve, por completo, o problema de excesso de informação.

Agentes de informação podem, por exemplo, montar um jornal de notícias personalizado. Basta que ele saibam as preferências do usuário, onde se encontram as informações, como capturá-las e onde colocá-las.

Os desafios enfrentados pelos agentes de informação, caso sejam estáticos, são os mesmos dos agentes de interface, sendo agentes de informação móveis, os desafios são os mesmos do agentes móveis .

Os **Agentes Reativos** são uma categoria que não possui modelos simbólicos do ambiente no qual estão inseridos. Eles simplesmente reagem a estados do ambiente.

Maes (citada por Nwana, 1996:26) destaca três idéias chaves relacionadas a agentes reativos. Primeiro, não se tem uma especificação *a priori* do comportamento do conjunto de agentes reativos. Segundo, refere-se à decomposição de tarefas. Um agente reativo é visto como um conjunto de módulos que opera autonomamente e é responsável por tarefas específicas. O modelo global não está presente em nenhum dos agentes. Terceiro, os agentes

reativos tendem a operar em representações de baixo nivel que são próximas a de sensores e de coletores de dados, ao contrário de representações simbólicas de alto nível, presentes em outros tipos de agentes.

Agentes reativos são considerados relativamente simples e fáceis de entender. Uma das justificativas para isso é que eles têm de aprender muito pouco. Eles não têm que efetuar planejamento ou revisar o modelo de seu ambiente, eles simplesmente reagem. E suas reações dependem do que acontece no momento atual.

Um dos benefícios que motivaram o trabalho com agentes reativos é a de que sejam mais robustos e tolerante a falhas que outros sistemas de agentes, devido, sobretudo, à sua simplicidade.

Dentre as áreas que podem se beneficiar deste tipo de agente pode-se citar: jogos e simuladores, onde os agentes interpretam elementos desses ambientes, onde cada agente reage a estímulos ou estados do ambiente no qual estão inseridos de uma maneira programada. Outra área é a de robôs, em especial os de brinquedo e os que realizam montagens, onde cada sensor ou motor pode ser controlado por um ou mais agentes.

Com relação aos agentes reativos, algumas questões não são óbvias. quantos agentes são necessários para se atingir determinado objetivo? Para um universo muito grande destes agentes, como descrever se atingiram ou não o objetivo? Também, não são desprezíveis os desafios para trabalhos nesta área: expansão do número e o alcance na utilização deste tipo de agente; desenvolvimento de metodologias que facilitem o desenvolvimento de tais agentes; escalabilidade e performance dos agentes.

Para se aproveitar melhor os benefícios e minimizar as deficiências, algumas aplicações mistas denominadas híbridas - **Agentes Híbridos** - podem vir a ser utilizadas. Ainda assim, não se pode afirmar que qualquer das aplicações híbridas existentes seja melhor do que outra, mesmo entre as cinco originais. Pode-se simplesmente detectar que, em determinada aplicação, pode vir a ser a mais adequada.

As aplicações, benefícios potenciais, assim como eventuais problemas a serem enfrentados no desenvolvimento desses cinco tipos de agentes: colaborativos, de interface, móveis, de informação/*Internet* e reativos não permite falar se um agente é melhor ou pior do que outros. Para cada aplicação um ou mais deles podem se mostrar mais adequados. Entretanto, mesmo o melhor em cada aplicação possui aspectos negativos.

Considera-se híbrida, uma abordagem que utiliza agentes de duas ou mais filosofias diferentes em um único agente. Podem ser colaborativa, de interface, móvel, de informação ou reativa.

Dentre as arquiteturas de agentes híbridos destacam-se:

A) **A arquitetura que casa as camadas deliberativas (colaborativa) com a reativa**, de autoria de Muller e outros (citado por Nwana,1996:30) do Centro de Pesquisa Alemão para Inteligência Artificial (Fig.4). Baseada em três camadas de controle, com a primeira relacionada ao comportamento (Behaviour-based Layer - BBL), a segunda ao planejamento local (Local Planning Layer - LPL) e a terceira ao planejamento cooperativo (Cooperative Planning Layer - CPL). A camada responsável pela eficiência, reatividade e robustez é implementada pela BBL que contém padrões de comportamento, ou seja, regras de ação frente a determinadas situações. Esta parte descreve as habilidades de reação dos agentes, o que é extremamente útil quando se enfrenta situações nas quais o tempo é crítico. A camada intermediária implementa comportamento direcionado ao cumprimento de objetivos locais, enquanto a camada superior possibilita ao agente cooperar com outros agentes para atingir objetivos de sistemas multi-agentes e resolver conflitos.

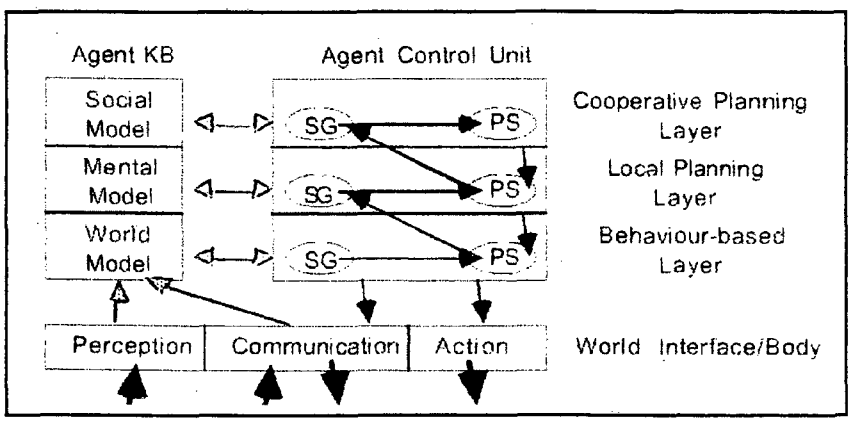


Figura 4 - Arquitetura híbrida (Nwana, 1996:30, adaptada de Fischer et alli., 1995).

Esta arquitetura trabalha também com outros modelos. As camadas CPL, LPL e CPL, por exemplo, podem operar em modelos sociais, mentais e de mundo respectivamente.

B) **A arquitetura integrada para agentes inteligentes** (*Adaptative Intelligent Systems* - AIS) que consiste de duas camadas: física e cognitiva. A primeira coordena percepção e ação, em sensores, interpretadores e filtros, etc. A segunda recebe informações da primeira para construir um modelo para interpretação, planejamento. Seu objetivo é prover

uma arquitetura para construir agentes inteligentes adaptativos que pode operar em um nicho especializado mas em constante mudança. Estas mudanças envolvem diferentes combinações de tarefas, configurações dos recursos disponíveis, condições contextuais que podem variar entre uma situação benigna e uma estressante, além de diferentes critérios de performance. Esta arquitetura foi implementada em uma unidade de monitoramento de terapia intensiva denominada Guardian (Hayes-Roth's, 1995) que se utiliza do conceito: o comportamento deflagrado por uma situação. Esta unidade é capaz de monitorar constantemente vinte variáveis de um paciente e ocasionalmente muitas outras. Este trabalho concluiu que:

- O nicho ocupado pelo AIS representa um incremento substancial aos requerimentos de comportamento comparado com nichos ocupados por típicos agentes de IA.
- Os nichos que podem ser ocupados pelo AIS aparentam representar objetivos alcançáveis. Não sobrecarrega os autores com a complexidade do comportamento humano, e sim apresentaram característica que ele tem - capacidade de adaptação.
- Os AIS representam aplicações reais e significativas, tais como: sistemas de monitoramento inteligentes; sistemas de vigilância inteligentes, sistemas inteligentes associados.

Agentes que apresentem estas características certamente têm real utilidade prática e social.

Sistemas de **Agentes Heterogêneos**, diferentemente dos sistemas híbridos descritos anteriormente, referem-se a um conjunto de dois ou mais tipos diferentes de agentes que pertencem a duas ou mais classes diferentes de agentes. Sistemas heterogêneos podem conter ainda um ou mais agentes híbridos.

A grande motivação para o desenvolvimento de sistemas de agentes heterogêneos vem da grande demanda por interoperabilidade. O argumento principal é que no mundo existem programas tão diversos quanto diversos são os domínios existentes. Contudo estes programas trabalham em sua maioria isolados, ou em pequenas associações. Genesereth & Ketchpel (1994) citam a interoperabilidade como um complemento de agregação de valor ao que estes programas já produzem isolados uns dos outros. Isto possibilita um grande salto na produtividade e no aproveitamento dos dados armazenados e/ou controlados por estes programas.

Um requisito chave para a interoperabilidade dos agentes é possuir uma linguagem na qual possam comunicar-se uns com os outros. Os benefícios potenciais em se ter uma tecnologia de agentes heterogênea são muitos. Pode-se citar:

- Aplicações isoladas podem agregar um valor maior através da interoperabilidade de sistemas heterogêneos.
- O problema de *software* legado pode ser aliviado porque pode livrar-se de ter que re-escrever o programa toda vez em que vá se fazer uma versão, podendo reduzir, também, as rotinas de manutenção, de atualizações ou correções.
- A engenharia de *software* baseada em agente oferece uma nova abordagem ao projeto de *software*, em particular quanto à interoperabilidade, tornando-se mais clara, à medida que novas metodologias e ferramentas forem desenvolvidas.

Genesereth & Ketchpel (1994) observaram que a engenharia de *software* é freqüentemente comparada à programação orientada a objeto, em que um agente como um objeto se utilizam de uma interface baseada em mensagens em sua estrutura interna de dados e algoritmos. Todavia, há uma diferença: em programação orientada a objeto o significado de uma mensagem pode ser diferente de objeto para objeto, em consonância com o polimorfismo; em sistemas baseados em agente usa-se normalmente uma linguagem comum com uma semântica independente dos agentes.

1.3 - Sistemas de agentes

Um sistema de agentes é uma plataforma que pode criar, interpretar, executar, transferir ou extinguir agentes. Assim como um agente, ao sistema de agentes é associada uma autoridade que identifica para qual pessoa ou organização o sistema de agentes trabalha. Um servidor pode conter um ou mais sistemas de agentes e um tipo de sistema de agente descreve o perfil de um agente. Por exemplo: Se o tipo sistema de agentes for o Aglet, o sistema de agentes foi implementado pela IBM, suporta Java como a linguagem dos agentes, utiliza-se de itinerário para deslocar-se e usa serialização através de objetos Java.

O MASIF reconhece tipos de sistemas de agentes que suportam múltiplas linguagens e métodos de serialização. Um cliente requisitando uma função de um sistema de agentes deve especificar o perfil do agente (tipo, linguagem e método de serialização).

Um agente possui a capacidade de transferir-se de um local para outro.

Um **local** é um contexto no qual um agente se executa, é associado a uma localização que consiste do nome do local e de um endereço do sistema de agentes de onde ele se encontra. Um sistema de agentes pode conter um ou mais locais e um local pode conter um ou mais agentes. Se um sistema de agentes não implementa um local, então o mesmo funciona com um local que pode ser definido como padrão. Este contexto provê funções, tais como controle de acesso. O local de origem e o de destino podem estar em um mesmo sistema de agentes, ou em diferentes sistemas de agentes.

Uma **região** é um conjunto de sistemas de agentes com a mesma autoridade, mas não necessariamente do mesmo tipo. Regiões permitem que mais de um sistema de agentes representem uma mesma pessoa ou organização. Uma região pode ter um conjunto de privilégios, que pode não ser concedido a outros agentes.

A **interconexão entre regiões** é feita através de uma ou mais redes, que pode, inclusive, partilhar um serviço de nomeação de agentes. Outros sistemas de agentes e clientes que estejam fora da região acessam a região através de sistemas de agentes que estão expostos ao mundo exterior. Estes sistemas que realizam a conexão com o mundo exterior são denominados como pontos de acesso da região.

Codebase especifica a localização das classes utilizadas pelo agente. Pode ser um sistema de agente ou um objeto não-CORBA. Caso um sistema de agentes seja o responsável pelo provimento das classes necessárias, o *codebase* deve possuir informação suficiente para localizar o sistema de agentes, tal sistema é denominado provedor de classes. É possível para um agente acessar um sistema de agentes mesmo que não consiga acesso direto , por exemplo no caso de utilização de *firewall*.

A **Infra-estrutura de comunicações** provê serviços de comunicação, nomeação e serviços de segurança para o sistema de agentes, normalmente RPC (*Remote Procedure Calling*). Comparativamente, para agentes fixos (estáticos) têm suas necessidades de comunicação atendidas pelos sistemas orientados à objeto usuais, tais como: CORBA, DCOM e RMI.

Localidade pode ser definida, no contexto de agentes móveis, como a proximidade com o agente de sistemas, no mesmo *host* ou na mesma rede.

Serialização é um processo de armazenamento de agente em uma forma serializada, suficiente para reconstruir o agente. Desserialização é o processo inverso. A forma serializada deve ser capaz de identificar e verificar as classes dos campos salvos.

1.3.1 - Interação entre Agentes

São definidos três tipos de interações entre agentes com relação à interoperabilidade: Criação remota de agentes, transferência de agentes e método de invocação de agentes.

A **criação remota de agentes** ocorre quando um programa cliente interage com o sistema de agente destinado, para solicitar a criação de um agente de uma classe em particular. Este cliente pode ser um outro agente ou não. O cliente realiza a autenticação de si mesmo para o sistema de agentes destinado, e estabelece a autoridade que o novo agente possuirá. Além de fornecer os argumentos para inicialização e caso necessário as classes para se instanciar e executar o agente.

Na **transferência de agentes** o local de destino e a qualidade do serviço de comunicação são identificados. Sendo que este último não é especificado pelo MASIF, que o deixa em aberto para os responsáveis pela implementação. Quando o sistema de agentes (do destino) concorda com a transferência, o estado do agente, sua autoridade, credenciais de segurança, e se necessário o código e demais classes para execução desse agente são transferidos a ele. O sistema de agentes reinstancia o agente e inicia sua execução.

A utilização de plataforma incorrerá em certos custos que incluem entre outros, o aumento da quantidade de memória requerida, e atrasos no acesso de cada participante. Entretanto, a linguagem Java, utilizada como base para muitas plataformas de agentes móveis, tem se apresentado bastante adequada.

O tamanho dos agentes depende da função. Para configuração e diagnóstico, eles podem tornar-se grandes, em razão de algoritmos ou ferramentas eventualmente complexas. Todavia, os agentes podem ampliar suas habilidades à medida que necessitem, carregando os recursos adicionais dependendo do ambiente local e suas necessidades.

Quando um agente **invoca um método** de outro agente ou objeto quando autorizado e possui uma referência ao objeto. A infra-estrutura de comunicações deve invocar o método indicado, ou retornar uma falha de indicação. Quando um agente invoca um método, a informação de segurança fornecida à infra-estrutura de comunicação deve ser o nível de autoridade do agente. Este método é suportado pelos sistemas atuais de objetos distribuídos.

O quadro 2 apresenta alguns benefícios potenciais do uso de agentes móveis.

Benefícios Possíveis	**Justificativa**
Eficiência	Ocupação da CPU é limitada, porque um agente móvel executa somente um nó por vez. Outros nós não executam um agente enquanto não é requerido.
Economia de espaço	Consumo de recursos é limitado, porque um agente ocupa somente um nó por vez. Já múltiplos servidores estáticos necessitam de duplicação e funcionamento em todos os locais. Objetos remotos provem benefícios semelhantes mas os custos de *middleware* podem ser altos.
Redução no tráfego da rede	Códigos são usualmente menores do que dados em processos, logo a transferência de agentes móveis até as fontes de dados cria menos tráfego do que transferência de dados. Objetos remotos podem ajudar em alguns casos mas o controle de parâmetros pode ser bem elevado.
Interação autônoma assíncrona	Agentes móveis podem ser delegados a executarem determinadas funções mesmo que a entidade que delega não permanece ativa.
Robustez e tolerância a falhas	Caso um sistema distribuído comece a apresentar mau funcionamento, então agentes móveis podem ser usados para incrementar a disponibilidade de certos serviços em certas áreas. Por exemplo: a densidade de agentes para detecção ou reparo pode ser aumentada. Algum nível de gerenciamento dos agentes é requerido para assegurar que um sistema baseado em agentes cumpra sua finalidade.
Suporte a ambientes heterogêneos	Agentes móveis são separados de seu servidores pela mobilidade da plataforma. Se a plataforma está no lugar, os agentes podem ir a qualquer sistema. Os custos de execução de uma máquina virtual Java em um mecanismo está decrescendo.
Extensibilidade de serviços *on-line*	Agentes móveis podem ser utilizados para aumentar a capabilidade de aplicações, por exemplo provendo serviços. Isto permite a construção de sistemas que são extremamente flexíveis.
Conveniente desenvolvimento de paradigma	A criação de sistemas distribuídos baseados em agentes é relativamente fácil. A parte difícil é a plataforma móvel, mas quando no lugar, torna a criação de aplicações mais fácil. Com o amadurecimento da área, aplicações de desenvolvimento rápidas, de alto nível, serão necessárias. É bem provável que as ferramentas para programação orientadas a objetos irá envolver-se com o desenvolvimento de ambientes orientados a agentes, que irão incluir alguma funcionalidade para facilitar a mobilidade do agente.
Fácil atualização de programas	Agentes móveis podem ser trocados a vontade. Em contraste, a alternância de funcionalidade dos servidores é complicada; especialmente se queremos manter um nível apropriado de qualidade de serviço.

Quadro 2 - Benefícios potenciais do uso de agentes móveis. (Bieszczad, 1998:2)

1.3.2 - Funções de um Sistema de Agentes

Quando da **transferência de um agente**, o agente móvel solicita ao sistema de origem uma transferência a outro sistema (destino) como parte de uma API interna. Quando o sistema de destino recebe a solicitação de transferência, ele executa o algoritmo descrito no item A da figura 9. Antes de um agente ser recebido no sistema de destino, este sistema deve determinar se ele pode interpretar este agente. Se ele pode interpretá-lo, então ele aceita o agente e executa o algoritmo apresentado no item B da figura 5.

A) Iniciando a transferência do agente (executado no lado da origem).
1. Suspender a execução do agente.
2. Identificar partes a transferir do agente.
3. Serializar a instância da classe e estado do agente.
4. Codificá-lo para o protocolo de transporte escolhido.
5. Autenticar o cliente.
6. Transferir o agente.

B) Recebendo um agente (executado no lado do destino)
1. Autenticar o cliente.
2. Decodificar o agente.
3. Desserializar a classe e o estado do agente.
4. Instanciar o agente.
5. Restaurar o estado do agente.
6. Iniciar a execução do agente.

Figura 5 - Algoritmos para a transferência de agentes (adaptada de Milojicic, 1998:631).

Existem três casos em que a transferência de classes é necessária:

- Instanciação do agente (criação remota do agente) - Quando um agente é criado remotamente pela solicitação de uma operação de criação no sistema de agente, a classe do agente é necessária para instanciá-lo. Se a classe não existe no destino, ela deve ser transferida a partir do sistema de origem.
- Instanciação do agente (transferência do agente) - Depois que um agente é transferido ao outro sistema, a classe do mesmo é necessária para instanciá-lo. Se ela não existir lá, é transferida a partir da origem.
- Execução do agente após a instanciação - Depois de um agente ser instanciado (conforme um dos dois modos anteriores), ele normalmente cria outros objetos. Se a classe desses objetos não estiver disponível no destino, ela é transferida a partir da origem.

O modelo conceitual é flexível o suficiente para suportar variações de **transferência de classes**, então para implementá-la tem-se mais de um método disponível. Especificamente o modelo suporta:

- Caso 1 - Transferência automática de classes - O sistema de origem (o provedor de classes ou o responsável pelo envio do agente) envia todas as classes necessárias à execução do agente a cada criação remota ou transferência de

agente. Isso elimina a necessidade de solicitação de classes pelo sistema de destino. Entretanto consome largura da banda desnecessariamente se o destino já possuir estas classes.

- Caso 2 - Transferência automática da classe do agente, transferência sob demanda de outras classes - O sistema de agentes da origem envia a classe necessária à instanciação do agente a cada criação remota de agente ou solicitação de transferência. Se mais classes forem necessárias após a instanciação do agente, o sistema destino emite uma solicitação ao provedor de classes. Se o provedor de classes não está acessível no destino, o sistema destino emite uma solicitação ao sistema emissor chamando o método *fetch_class* com o *codebase* como parâmetro. O emissor localiza a classe solicitada usando a informação do *codebase*, ou enviando uma solicitação a outro sistema de agente associado ao *codebase*. O sistema emissor pode ter colocado as classes na memória *cache*. Esta abordagem não requer que a origem determine todas as possibilidade necessárias a criação ou transferência de agentes, e é mais eficiente quanto mais classes estiverem disponíveis na memória *cache* do destino. Entretanto, a criação ou transferência pode falhar se o sistema de destino não puder acessar o sistema de origem para transferir as classes necessárias. Por exemplo: Quando o sistema de origem for um computador portátil que desconectou-se após a criação ou envio do agente.
- Variações dos dois itens anteriores - a) Caso 2 quando transferindo um agente e caso 1 quando criando um agente remotamente. Quando a criação remota de agentes é iniciada por um cliente que não está sempre conectado, todas as classes são transferidas automaticamente para permitir as operações de criação do agente; b) A origem lista todas as classes necessárias para executar as operações específicas da criação do agente remotamente. O destino solicita então somente as classes que não estão em sua *cache*. Esta abordagem é bem eficiente, entretanto requer que o sistema de origem saiba quais classes o agente necessitará antes de efetuar a solicitação de criação ou envio de agentes.

Para **criar um agente**, um sistema de agente cria uma instância de uma classe de agente em um local padrão ou determinado pelo cliente. A classe do agente especifica a interface e a implementação do agente. O agente se executa. Cada agente executa sua tarefa em

separado e independentemente de outros agentes, o que é denominado *thread*. Um algoritmo simplificado é apresentado na figura 6.

1. Iniciar um agente.
2. Instanciar a classe do agente.
3. Gerar um nome único global para o agente.
4. Iniciar a execução do agente

Figura 6 - Algoritmo para a criação de agentes (Adaptada de Milojicic, 1998:632).

Um sistema de agentes deve gerar um nome único para si, seus agentes e locais que cria, através de um serviço de nomeação de agentes. Este nome deve ser único em todos os sistemas de agentes onde o agente possa ir.

Quando um agente necessita comunicar-se com outro, o mesmo deve saber qual o sistema de destino para estabelecer uma comunicação. A habilidade em localizar um agente móvel em particular é, também, útil no gerenciamento de agentes.

2 - O MODELO *OMG MASIF*

O MASIF[2] trata da interoperabilidade de sistemas de agentes móveis escritos em uma mesma linguagem, inclusive de diferentes fabricantes, e não entre linguagens. Além disso, o MASIF não padroniza operações em agentes locais tais como: interpretação, serialização e execução. Com a idéia de assegurar a interoperabilidade, as interfaces têm sido definidas ao nível de sistema de agente ao invés de ser ao nível de agente. Levando isso em conta, têm-se padronizado pelo MASIF (Fig.7): gerenciamento de agentes; transferência de agentes; nomeação de agentes e de sistema de agentes; e sintaxe de tipo e de localização de sistema de agentes.

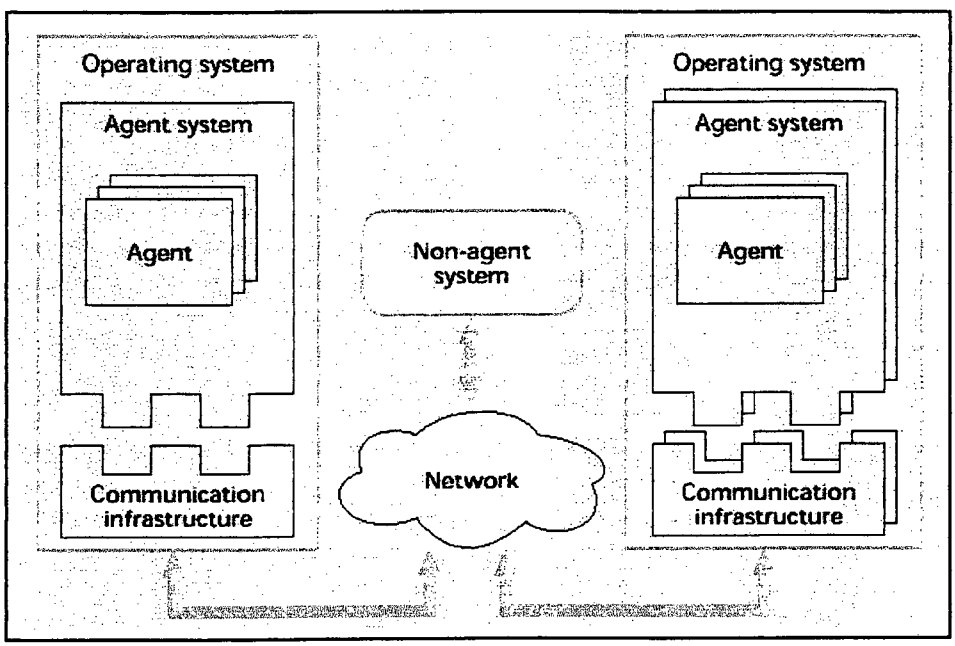


Figura 7 - Arquitetura MAF[3] (*Mobile Agent Facility*).

Gerenciamento de agentes - Um administrador de sistemas gerencia sistemas de agentes de diferentes tipos, via operações de uma maneira padronizada: cria um agente, suspende sua execução, inicializa-o e o extingue (Fig.8).

---

[2] Proposto originalmente por GMD Fokus e IBM, apoiados por Crystaliz e General Magic em novembro de 1997. Adotado como padrão OMG em fevereiro de 1998.

[3] A acrônimo MAF (*Mobile Agent Facility*), posteriormente transformada em MASIF (*Mobile Agent System Interoperability Facility*),origina-se da proposta original. Todavia algumas partes de especificação ainda a utilizam.

```
Interface MAFAgentSystem {

        Name create_agent(in Name agent_name, ...) Raises (...);
        void resume_agent(in Name agent_name) raises (...);
        void suspend_agent(in Name agent_name) raises (...);
        void terminate_agent(in Name agent_name) raises (...);

        void terminate_agent_system( ) raises (...);

        AgentStatus get_agent_status(in Name agent_name) raises (...);
        AgentSystemInfo get_agent_system_info( );
        AuthInfo get_authinfo(in Name agent_name) raises (NameInvalid);
        NameList list_all_agents( );
        NameList list_all_agents_of_authority(in Authority authority);
        Locations list_all_places ( );
}
```

Figura 8 - Operações MAF de gerenciamento.

Transferência de agentes - É desejável que aplicativos de agentes possam mover-se livremente entre diferentes sistemas de agentes, o que resulta em uma infra-estrutura comum e uma maior base de sistemas às quais o agente pode acessar (figura 9).

```
Interface MAFAgentSystem {
                ...
        OctetStrings fetch_class (...) Raises (...);
        void receive_agent (...) Raises (...);
                ...
}
```

Figura 9 - Operações MAF para transferência de agente.

Nomeação de agentes e de sistema de agentes - Padroniza a sintaxe e a semântica de agentes e de sistemas de agentes, o que lhes permite identificarem-se mutuamente, assim como a clientes identificarem agentes e sistemas de agentes.

Sintaxe de tipo e de localização de sistema de agentes - A transferência de agentes não pode ser completada se o tipo de sistema de agente não suporta o agente. A sintaxe de localização é padronizada para que os sistemas de agentes possam localizar-se mutuamente (Fig.10).

```
interface MAFFinder {

        void register_agent (...) raises (NameInvalid);
        void register_agent_system (...) raises ( NameInvalid);
        void register_place (...) raises (NameInvalid);

        Locations lookup_agent (...) raises (EntryNotFound);
        Locations lookup_agent_system (...)  raises (EntryNotFound);
        Locations lookup_place (place_name) raises ( EntryNotFound);
};

Interface MAFAgentSystem {
        Location find_nearby_agent_system_of_profile (profile) raises (...);
                        ...
```

Figura 10 - Operações MAF para rastreamento de agente.

O quadro 3, a seguir, descreve os tipos de interoperabilidade especificada no MASIF e estima a complexidade requerida dos sistemas de agentes para suportá-la. A comunicação de agentes está fora do escopo do MASIF e é abordada pelo CORBA.

Função	Especificada no MASIF	Complexidade
Gerenciamento de agente	Sim	Não
Rastreamento de agentes	Sim	Direta
Comunicação de agentes	Não	Não se aplica
Transporte de agentes	Sim	Complexa

Quadro 3 - Interoperabilidade especificada pelo MASIF e a complexidade associada.

2.1 - MAF IDL

O MAF (Mobile Agent Facility) é um conjunto de definições e interfaces que permitem uma interoperabilidade entre sistemas de agentes móveis (Fig.11). Foi elaborado de uma maneira simples e genérica de modo a permitir desenvolvimentos futuros em sistemas de agentes móveis. Contém duas interfaces: MAFAgentSystem - que define operações com agentes do tipo criar, transferir, receber, suspender (pausa) e extinguí-lo; e MAFFinder - que define operações de localização de agentes, locais e sistemas de agentes.

Estas interfaces foram definidas ao nível de sistemas de agentes, ao invés de ao nível de agentes por questões relativas a interoperabilidade. Foi discutido que tanto os sistemas de agentes quanto os agentes podem ser objetos CORBA ou não. Como todo agente está em um sistema de agentes, sua implementação (agente) depende da implementação do sistema no

qual está inserido. Estas implementações devem levar em conta a linguagem, o método de serialização e mecanismos de autenticação.

Além disso, as operações definidas no MAF relativas à criação, transferência, pausa, e extinção formam, segundo a proposta MASIF, um conjunto básico de operações suficientes para a migração de agentes de um sistema a outro. Logo, não haveria necessidade de se definir tais operações nos agentes.

> *Os serviços CORBA foram projetados inicialmente tendo-se em vista objetos estáticos. Em se tratando de agentes móveis alguns serviços CORBA não funcionam tão bem. Como por exemplo, os serviços de nomeação. Já interface MAFFinder funciona como uma interface de uma base de dados dinâmica de nomes, locais e sistemas.* (Milojicic, 1998:637)

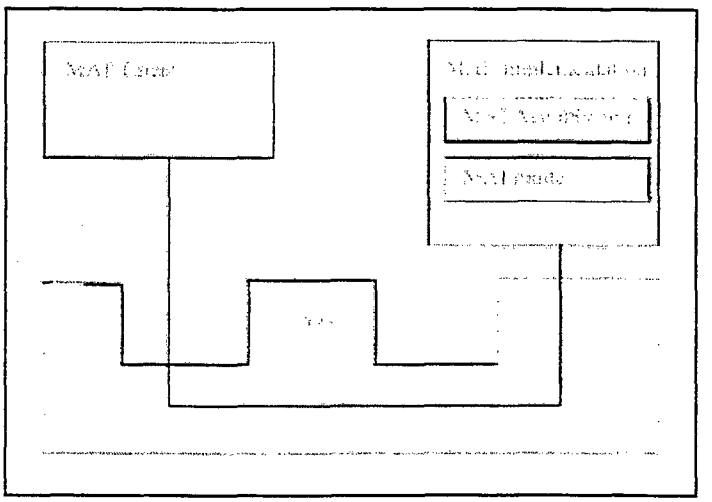


Figura 11 - Relação entre a Interface MAF e o ORB.

2.1.1 - Definições e estruturas do MAF

No módulo MAF, a estrutura ***Name*** (nome), contém três atributos (Fig.12):

- *authority* - identifica a pessoa ou organização que gerou o agente, no CORBA é utilizado o termo *principal*;
- *identity* - identificação do agente;
- *agent_system_type* - tipo do sistema de agentes que gerou este agente.

A identificação de tipo de agente (agent_system_type) se faz necessária, pois é possível que uma pessoa ou organização possua mais de um tipo de sistema de agentes que gerem valores iguais para os atributos que identificam a origem e os agentes. O Aplicativo cliente vale-se então da identificação do tipo e de sistema para distinguí-los.

```
typedef OctetString Authority;
typedef OctetString Identity;

struct Name{
    Authority           authority;
    Identity            identity;
    AgentSystemType     agent system type;
};
typedef string Location;
```

Figura 12 - Estrutura *Name* e seus atributos.

No módulo MAF, a estrutura ***Class Name*** (Nome de Classe) define a sintaxe para os nomes de classe e uma string para identificar, unicamente uma classe, dentro de um determinado escopo, que é o sistema de agentes. A especificação MAF não prevê um modo de assegurar um nome único para os agentes em um ambiente mais amplo do que este (Fig.13).

A classe *ClassOne* do sistema A deve ser única dentro do sistema A. Analogamente a classe *ClassOne* do sistema B deve ser única no escopo do sistema B. Note que quando o sistema C solicita a classe *ClassOne* do sistema A e após isso requisita uma outra classe *ClassOne* do sistema B, o sistema C deve poder distinguí-los. Para fins didáticos, foram utilizados duas formas geométricas para diferenciá-las.

Quando o sistema de agentes C deseja criar um agente no sistema D e a classe *ClassOne* estiver envolvida, o sistema C utiliza o parametro *Create_Agent* para especificar ao sistema D qual *ClassOne* é necessária.

Conforme observado na proposta MASIF, isso pode gerar transferências desnecessárias de classes. Considere que em outra ocasião o sistema D necessite instanciar uma classe do tipo da *ClassOne* do sistema A. O sistema D pode então não reconhecer que se trata da mesma classe recebida anteriormente de C. Ainda conforme observado na proposta MASIF, caso os administradores das regiões concordem em desenvolver um esquema conjunto de nomeação de agentes, a situação descrita anteriormente, na qual o sistema C possui dois agentes com o mesmo nome, não ocorreria.

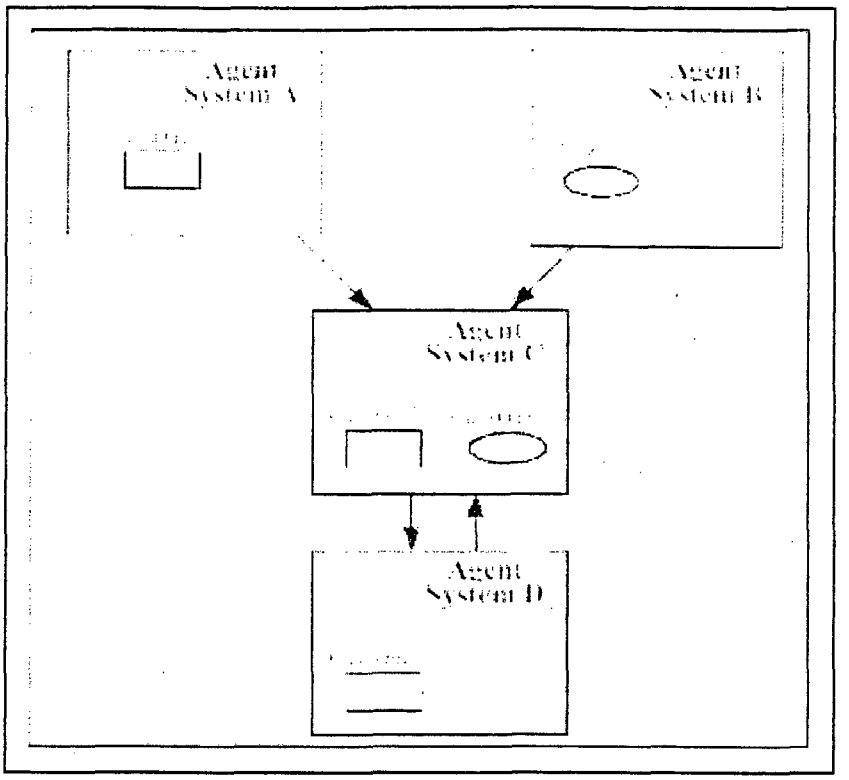


Figura 13 - Nome único de classe em cada sistema de agentes.

No módulo MAFFinder, a *string* ***Location*** especifica o caminho até um sistema de agentes baseado no nome do sistema de agentes, agente, ou local. A string pode conter uma das seguintes formas: uma URI (*Uniform Resource Identifier*) contendo um nome CORBA e independe de protocolos; ou uma URL (*Uniforn. Resource Locator*) contendo um endereço *Internet*, que é mais adequado a agentes móveis e a *Internet*.

Para determinação do formato da *string Location*, o aplicativo (cliente) identifica o que vem após o dois-pontos (:). Caso seja "*CosName*" trata-se de um nome CORBA, caso seja "mafiiop" trata-se de um endereço *Internet*.

2.1.2 - Identificadores de autoridade de nomeação

Os identificadores atribuídos aos parâmetros tais como tipo identificador de linguagem, tipo de sistema de agente, autenticador e serialização devem ser únicos em quaisquer implementações do MAF. Para isso a atribuição de nomeação na área de agentes móveis é feita pela OMG (Fig.14), o que garante as vantagens da interoperabilidade entre diferentes aplicações MAF.

```
typedef short LanguageID;
typedef short AgentSystemType;
typedef short Authenticator;
typedef short SerializationID;
typedef sequence<SerializationID> SerializationIDList;
```

Figura 14 - Parâmetros cujos valores são gerenciados pelo OMG (MASIF cap 3 p. 9).

Como valores iniciais definidos pela OMG, pode-se citar:

- *LanguageID* - Java, Tcl, Perl, Scheme.
- *AgentSystemType* - Aglets, AgentTcl, MOA.
- *Authenticator - none, one-hop.*
- *SerializationID - Java Object Serialization.*

Cada um dos parâmetros acima teve seus valores sugeridos pela proposta MASIF à OMG, baseado nos tipos de sistemas, linguagens, mecanismos de codificação e autenticação em uso. São eles:

- LanguageID - Não especificado (0), Java (1), Tcl (2), Scheme (3), Perl (4).
- AgentSystemType - Sistema não-agente (0), Aglets (1), MOA (2), AgentTcl (3).
- Encoding (Authenticator) Mechanisms - nenhum (1), one-hop (2).
- SerializationID - Não especificada (0), Java Object Serialization (1).

2.2 - *MAFAgentSystem*

A interface *MAFAgentSystem* define métodos e objetos para "manusear" os agentes, tais como: criar, transferir, ativar sua execução, pausá-lo, extinguí-lo, chamá-lo de volta, dentre outros. Estes métodos constituem um conjunto de operações para a transferência dos agentes, e serão abordados um a um adiante.

Create_agent ( )

Utilizada pelo sistema de agente para criação de um agente quando solicitado. Retorna o nome do agente criado. Mesmo um cliente sem as capacidades de um agente podem

usar este método. E o agente de destino é livre para responder a solicitação ou repassá-la a outro sistema na mesma região. Um modelo de sintaxe é mostrado na figura 15:

```
Name create_agent (in Name agent_name, in AgentProfile agent_profile, in
        OctetString agent, in String place_name, in Arguments arguments, in
        ClassNameList class_names, in String code_base, in MAFAgentSystem
        class_provider) raises (ClassUnknown, ArgumentInvalid,
        DeserializationFailed, MAFExtendedException);
```

Figura 15 - Sintaxe de create_agent ( ). (MASIF cap 3 p. 12).

Como parâmetros contém:

- *agent_name* - Valor retornado como nome do agente. O autor do agente é o programa cliente que solicita sua criação. Este autor pode ser outro sistema de agentes.
- *agent_profile* - Contém informações sobre o agente, tais como tipo do sistema de agente utilizado na sua criação e o método usado na serialização.
- *agent* - Parâmetro para uso geral com qualquer informação não prevista nos outros itens. A única restrição é a de que o sistema de agente que irá cria o agente deve ser capaz de entender o conteúdo deste parâmetro. Conforme MASIF pode ser inclusive definições de classes, de estado do agente e de classes necessárias para instanciação do agente.
- *place_name* - Local onde o agente criado deve ser colocado. Caso não seja especificado será utilizado o local padrão do sistema.
- *arguments* - Especifica os argumentos para o construtor do agente .
- *class_names* - Contém a lista de todas as classes necessárias para instanciação do agente, pode inclusive não listar nada.
- *code_base* - Contém referência ao *codebase* contendo as definições de classe necessárias. Se necessário é retornado ao cliente via *fetch_class*. Somente a *class_provider* necessita saber a sintaxe desta string se a fonte destas definições de classe for um agente de sistema. É possível ao sistema de destino recuperar as definições de classe diretamente usando esta informação no *codebase*.
- *class_provider* - referências a fonte do cliente que provê as definições de classe necessárias.

As exceções podem ser:

- *ClassUnknown* - falha em encontrar a definição de classe;
- *ArgumentsInvalid* - argumentos passados não combinam com nenhuma assinatura de construtor de agente;
- *DeserializationFailed* - o sistema não pode instanciar um agente porque não pode decodificar o *OctetString* do agente.
- *MAFExtendedException* - erro para uso geral, quando o erro não se enquadra em nenhuma das exceções anteriores.

Fetch_class ( )

Utilizada pelo sistema de agente para retornar as definições de uma ou mais classes. No caso de um sistema de agente não orientado a objeto é usado para localizar e trazer algum código. Este método é usado para buscar um *codebase* e cliente especificado. O sistema de agente que recebeu a solicitação pode saber se ele é ou não o provedor de classe examinando o *codebase* recebido.

Como uma implementação opcional, caso este sistema não seja o provedor de classes requerido, pode então retornar ou repassar as classes recebidas.

Um modelo de sintaxe é mostrado na figura 16:

```
OctetStrings fetch_class (in ClassNameList class_name_list, in string code_base, in
        AgentProfile agent_profile) raises (ClassUnknown,
        MAFExtendedException);
```

Figura 16 - Sintaxe de *fetch_class ( )* (MASIF cap 3 p. 14).

Como parâmetros contém:

- *class_name_list* - contém o nome das definições de classe requeridas.
- *code_base* - Contém referência ao *codebase* contendo as definições de classe necessárias. A sintaxe deste parâmetro pode variar de um sistema de agentes para outro sem afetar a interoperabilidade. O cliente provê o *code_base* via *create_agent ( )* ou *receive_agent ( )*, e é retornado ao cliente via *fetch_class*. Somente o provedor de classes necessita saber a sintaxe desta *string*.

- *agent_profile* - contém informação sobre a linguagem e método de serialização usado para o contexto de *create_agent ( )* ou *receive_agent ( )*.

As exceções podem ser:

- *ClassUnknown* - falha em encontrar a definição de classe.
- *MAFExtendedException* -erro para uso geral, quando o erro não se enquadra em nenhuma das exceções anteriores.

Receive_agent ( )

Usado por um sistema de agentes para instanciar e receber um agente. Um possível algoritmo para esta implementação poderia conter: Uma verificação se as classes necessárias para a instanciação do agente estão incluídas no parâmetro *receive_agent ( )*; Uma chamada *fetch_class ( )* para buscar se necessário as classes não disponíveis; Deserialização do agente efetuada no local solicitado ou no local padrão caso não haja nenhuma citação ao local. Um modelo de sintaxe é mostrado na figura 17:

```
void receive_agent (in Name agent_name, in AgentProfile agent_profile, in
        OctetStrings agent, in string place_name, in ClassNameList class_names, in
        string code_base, in MAFAgentSystem agent_sender) raises
        (ClassUnknown, DeserializationFailed, MAFExtendedException);
```

Figura 17 - Sintaxe de *receive_agent* ( ) (MASIF cap 3 p. 20).

Como parâmetros contém:

- *agent_name* - identifica únicamente um agente, e inclui identificação da pessoa ou organização representada pelo agente .
- *agent_profile* - contém informações sobre o agente, tais como: tipo do sistema de agente que o criou e método de serialização usado.
- *agent* -Parâmetro para uso geral com qualquer informação não prevista nos outros itens. O sistema de agentes que recebe o agente deve ser capaz de entender o conteúdo deste parâmetro. Conforme MASIF pode ser inclusive definições de classes, de estado do agente e de classes necessárias para instanciação do agente. As definições de classe incluídas neste parâmetro afetam o parâmetro *class_names*. Caso uma classe seja incluída neste item, não é

necessário incluí-lo na lista *class_names*. Analogamente, se uma classe for incluída na lista *class_names*, ela não deve ser incluída neste item.

- *place_name* -Local onde o agente deve ser colocado. Caso não seja especificado será utilizado o local padrão do sistema.
- *class_names* -Contém a lista de todas as classes necessárias para instanciação do agente, pode inclusive não listar nada. Como citado anterioremente no parâmetro *agent*, as definições de classe listadas aqui não necessitam ser explicitadas em *agent* e vice-versa.
- *code_base* - Contém referência ao *codebase* contendo as definições de classe necessárias. É retornado ao cliente via *fetch_class*. Somente o cliente necessita saber a sintaxe desta string. A sintaxe deste parâmetro pode variar de um sistema de agentes para outro sem afetar a interoperabilidade.
- *agent_sender* - Contém a referência ao sistema de agentes que esta iniciando a transferência.

As exceções podem ser:

- *ClassUnknown* - falha em encontrar a definição de classe.
- *DeserializationFailed* - o sistema não pode instanciar um agente porque não pode decodificar o *OctetString* do agente.
- *MAFExtendedException* - Erro para uso geral, quando o erro não se enquadra em nenhuma das exceções anteriores.

Resume_agent ( )

O método *resume_agent ( )* reativa a execução de um agente que foi suspenso (pausado) pela função *suspend_agent ( )*. Um modelo de sintaxe é mostrado na figura 18:

```
void resume_agent (in Name agent_name) raises (AgentNotFound, Resume Failed,
        AgentsIsRunning);
```

Figura 18 - Sintaxe de *resume_agent* ( ) (MASIF cap 3 p. 22).

Como parâmetros contém:

- *agent_name* - identifica o agente a ser reiniciado.

As exceções podem ser:

- *AgentNotFound* - O sistema de agentes não encontrou o agente.
- *ResumeFailed* - O sistema não conseguiu reiniciar o agente.
- *AgentIsRunning* - O agente já está em execução, não necessitando ser reativado.

Suspend_agent ( )

O método *suspend_agent ( )* suspende a execução de um agente especificado. A execução deste agente pode ser reativada por meio da função *resume_agent ( )*. Para que seja reativado a execução do agente deve ser suspensa, mas seus estados devem ser mantidos. Um modelo de sintaxe é mostrado na figura 19:

```
void suspend_agent (in Name agent_name) raises (AgentNotFound,
        SuspendFailed, AgentsIsSuspended);
```

Figura 19 - Sintaxe de *suspend_agent* ( ) (MASIF cap 3 p. 23).

Como parâmetros contém:

- *agent_name* - identifica o agente a ser suspenso.

As exceções podem ser:

- *AgentNotFound* - O sistema de agentes não encontrou o agente.
- *ResumeFailed* - O sistema não conseguiu suspender a execução do agente.
- *AgentIsSuspended* - O agente já está com sua execução suspensa.

Terminate_agent ( )

O método *terminate_agent ( )* pára permanentemente a execução de um agente especificado. O agente fica impossibilitado de ter sua execução reativada. Um modelo de sintaxe é mostrado na figura 20:

```
void terminate_agent (in Name agent_name) raises (AgentNotFound,
        TerminateFailed);
```

Figura 20 - Sintaxe de *terminate_agent* ( ) (MASIF cap 3 p. 24).

Como parâmetros contém:

- *agent_name* - identifica o agente a ser parado.

As exceções podem ser:

- *AgentNotFound* - O sistema de agentes não encontrou o agente.
- *TerminateFailed* - O sistema não conseguiu parar a execução do agente.

Terminate_agent_system ( )

O método *terminate_agent_system ( )* pára permanentemente a execução do sistema de agentes especificado. Dependendo da implementação, o sistema de agentes pode informar a todos os agentes de que será finalizado. Um modelo de sintaxe é mostrado na figura 21:

```
void terminate_agent_system ( ) raises (TerminateFailed);
```

Figura 21 - Sintaxe de *terminate_agent*_system ( ) (MASIF cap 3 p. 24).

Este método não contém parâmetros, e sua exceção pode ser:

- *TerminateFailed* - O sistema não conseguiu parar a execução do sistema de agentes.

Find_nearby_agent_system_of_profile ( )

O método *find_nearby_agent_system_of_profile ( )* solicita ao MAFFinder que procure um sistema de agentes próximo que possa executar o agente que o cliente deseja enviar. Este é um método do MAFAgentSystem que confia ao MAFFinder a tarefa de localizar um sistema de agentes próximo do tipo correto.

Pode ser usado quando um agente deseja se comunicar com um sistema de agentes de um tipo diferente. Uma vez que a comunicação direta com este sistema não é possível, este método é utilizado para localizar um sistema compatível que esteja próximo ao sistema com o

qual o agente deseja se comunicar. Este sistema compatível atua como um intermediário no processo de comunicação. Um modelo de sintaxe é mostrado na figura 22:

```
Location find_nearby_agent_system_of_profile ( ) (in AgentProfile profile) raises
        (EntryNotFound);
```

Figura 22 - Sintaxe de *find_nearby_agent*_system_of_profile ( ) (MASIF cap 3 p. 15).

Como parâmetro contém:

- *profile* - identifica o tipo do agente enviado.

As exceção pode ser:

- *EntryNotFound* - O agente especificado não esta na lista de agentes do sistema de agentes atual.

Get_agent_status ( )

O método *get_agent_status ( )* retorna o estado atual de um agente especificado. Este estado é uma dentre três opções: *running* - o referido agente encontra-se em execução; *suspended* - o agente encontra-se com sua execução suspensa; *terminated* - o agente foi finalizado.

Um modelo de sintaxe é mostrado na figura 23:

```
Agent_Status get_agent_status (in Name agent_name) raises (AgentNotFound);
```

Figura 23 - Sintaxe de get_agent_status ( ) (MASIF cap 3 p. 16).

Como parâmetros contém:

- *agent_name* - identifica o agente cujo status se deseja saber.

As exceções podem ser:

- *AgentNotFound* - O sistema de agentes não encontrou o agente especificado.

Get_agent_system_info ( )

O método *get_agent_system_info ( )* retorna informações sobre o sistema de agentes tais como:

- *system_name* - nome do sistema de agentes;
- *system_type* - tipo do sistema de agentes. Por exemplo, Aglets, MOA ou AgentTcl;
- *language_maps* - a linguagem de programação que o sistema de agentes suporta. Por exemplo, Java, Tcl, Perl ou Scheme. Além do método de serialização, por exemplo, *JavaObjectSerialization*, ASN1_BER, ASN1_DER).
- *system_description* - breve descrição sobre o sistema de agentes. Esta descrição não é padronizada e depende da implementação.
- *major_version* - informação de versão sobre a implementação do sistema de agentes. Utilizado pela JVM (*Java Virtual Machine*) para verificar a máxima versão de classes com que ele pode lidar, rejeitando classes com versões posteriores.
- *minor_version* - informação de versão sobre a implementação do sistema de agentes. *get_authinfo ( )*.
- *serializations* - identifica o método de serialização usado pelo sistema de agentes. Por exemplo: *JavaObjectSerialization*, ASN1_BER, ASN1_DER.

Um modelo de sintaxe é mostrado na figura 24 a seguir:

```
AgentSystemInfo get_agent_system_info ( );
```

Figura 24 - Sintaxe de get_agent_system_info ( ) (MASIF cap 3 p. 17).

Este método não contém parâmetros, e retorna informações sobre o sistema de agentes. E não são previstas exceções.

Get_auth_info ( )

O método *get_authinfo ( )* informa se o agente foi autenticado e qual método teria sido utilizado. Caso haja a necessidade de uma maior segurança, o cliente pode autenticar o

sistema do agente antes de chamar este método. Um modelo de sintaxe é mostrado na figura 25:

```
AuthInfo get_authinfo (in Name agent_name) raises (AgentNotFound);
```

Figura 25 - Sintaxe de *get_authinfo ( )* (MASIF cap 3 p. 18).

Como parâmetros contém:

- *agent_name* - identifica o agente cuja informação se deseja saber.

As exceções podem ser:

- *AgentNotFound* - O sistema de agentes não encontrou o agente especificado.

Get_MAFFinder ( )

O método *get_MAFFinder ( )* retorna uma referência para um MAFFinder da região atual. Uma vez conseguida a referência ao MAFFinder, seus métodos podem ser utilizados para a localização de agente, locais e sistemas de agentes. Um modelo de sintaxe é mostrado na figura 26:

```
AuthInfo get_MAFFinder ( ) raises (MAFFinderNotFound);
```

Figura 26 - Sintaxe de *get_MAFFinder ( )* (MASIF cap 3 p. 18).

Este método não contém nenhum parâmetro. Apresenta como exceção:

- *MAFFinderNotFound* - O MAFFinder desta região não foi encontrado.

List_all_agents ( )

O método *list_all_agents ( )* lista todos os agentes registrados no sistema de agentes atual. Um modelo de sintaxe é mostrado na figura 27:

```
NameList list_all_agents ( );
```

Figura 27 - Sintaxe de *list_all_agents ( )* (MASIF cap 3 p. 19).

List_all_agents_of_authority ( )

O método *list_all_agents_of_authority ( )* é uma operação de gerenciamento que lista todos os agentes do sistema representados pela pessoa ou organização especificada. Um modelo de sintaxe é mostrado na figura 28:

```
NameList list_all_agents_of_authority (in Authority authority);
```

Figura 28 - Sintaxe de *list_all_agents_of_authority ( )* (MASIF cap 3 p. 19).

Este método possui parâmetro:

- *authority* - Pessoa ou organização cujos agentes se deseja listar.

Não há exceções para este método.

List_all_places ( )

O método *list_all_places ( )* é uma operação de gerenciamento que lista todos os locais do sistema de agentes atual registrados no *MAFFinder*. Um modelo de sintaxe é mostrado na figura 29:

```
Locations list_all_places ( );
```

Figura 29 - Sintaxe de *list_all_places ( )* (MASIF cap 3 p. 20).

Este método não possui parâmetros ou exceções.

2.3 - *MAFFinder*

A interface *MAFFinder* define métodos para a manutenção de um banco de dados para localização de agentes (*agent*), locais (*place*) e sistemas de agentes (*agent_system*). Podendo-se chamar, para cada um destes elementos, três métodos tais como: registro (*register*), busca (*lookup*) e cancelamento (*unregister*).

Por exemplo: Junto ao MAFFinder o agente pode ser registrado, procurado ou ter seu registro cancelado. O mesmo acontecendo com o local e com o sistema de agentes.

São quatro as possibilidades de se localizar um agente:

- Força bruta - São localizados todos os sistemas de agente da região e enviado um outro agente que percorrerá todos os sistemas em busca do agente que se dejeja localizar.
- Consulta a registros - Um agente ao deixar um determinado sistema de agentes deixa um registro no mesmo informando seu destino. Cada sistema de agentes tem seu próprio banco de dados de destino dos agentes. Para se localizar determinado agente deve-se consultar os registros de cada sistema por onde tenha passado, descobrindo-se assim para onde o mesmo prosseguiu.
- Consulta a um registro centralizado - Um agente ao deixar um sistema de agentes informa a um banco de dados centralizado informando seu destino. Cada região possui seu próprio banco de dados de localização de cada agente. Esta modalidade apresenta a desvantagem de adicionar um *overhead* ao processo de migração do agente, além de um possível gargalo em operções com este banco de dados.
- Divulgação - Somente existem registros dos locais. A localização de um agente somente é registrada se o mesmo divulgar seu local atual. Caso o agente não divulgue esta informação ele somente poderá ser localizado por força bruta ou consulta aos registros de cada sistema de agentes em uma região.

Lookup_agent ( )

O método *lookup_agent ( )* retorna a localização do agente especificado. O agente pode ser localizado pelo nome perfil. Este método pode ser chamado por um aplicativo cliente ou por outro agente. Um agente, ou um conjunto, pode ser localizado tanto pelo nome (parâmetro agent_name) quanto pelo perfil (agent_profile).

A busca pelo nome pode ser feita de muitas maneiras diferentes, já que depende da implementação a qual pode organizar os nomes de famílias e gerações de agentes permitindo a busca por membros de uma determinada família ou geração.

Pelo perfil a busca pode ser feita por qualquer uma das informações disponíveis no *agent_profile*.

Um modelo de sintaxe é mostrado na figura 30:

```
Locations lookup_agent ( in Name agent_name, in AgentProfile agent_profile) raises
        (EntryNotFound);
```

Figura 30 - Sintaxe de *lookup_agent* ( ). (MASIF cap 3 p. 26).

Como parâmetros contém:

- *agent_name* - Nome do agente a ser localizado.
- *agent_profile* - As informações que podem ser utilizadas são a linguagem de implementação, o tipo de sistema de agentes, o tipo de autenticação e o método de serialização.

A exceção pode ser:

- *EntryNotFound* - Não foi encontrado nenhum agente com estas informações;

Lookup_agent_system ( )

O método *lookup_agent_system ( )* retorna a localização de um sistema de agentes. A busca pode ser efetuada por nome ou por qualquer combinação possível dos itens do perfil do sistema (*agent_system_info*). Um modelo de sintaxe é mostrado na figura 31:

```
Locations lookup_agent_system (in Name agent_system_name, in AgentSystemInfo
        agent_system_info) raises (EntryNotFound);
```

Figura 31 - Sintaxe de *lookup_agent*_system ( ). (MASIF cap 3 p. 27).

Como parâmetros contém:

- *agent_system_name* - Nome do sistema de agentes a ser localizado.
- *Agent_system_info* - As informações disponíveis disponíveis para utilização são *system_name*, *system_type*, *language_maps*, *system_description*, *major_version*, *minor_version*, e *properties* as quais foram descritas anteriormente.

A exceção pode ser:

- *EntryNotFound* - Não foi encontrado nenhum sistema de agentes com estas características;

Lookup_place ( )

O método *lookup_place ( )* retorna a localização do local especificado, desde que registrado no MAFFinder. Utilizado quando o cliente possui somente o nome do local e deseja obter sua localização para envio de um agente. Um modelo de sintaxe é mostrado na figura 32:

```
Location lookup_place (in string place_name) raises (EntryNotFound);
```

Figura 32 - Sintaxe de *lookup_place* ( ). (MASIF cap 3 p. 28).

Como parâmetro contém:

- *place_name* - Nome do local a ser localizado.

A exceção pode ser:

- *EntryNotFound* - O local especificado não esta registrado no MAFFinder.

Register_agent ( )

O método *register_agent ( )* adiciona o nome do agente especificado na lista do MAFFinder. Caso este método seja chamado sobre um agente já registrado, o novo registro sobrepõe-se ao anterior. Um modelo de sintaxe é mostrado na figura 33:

```
void register_agent (in Name agent_name, in Location agent_location, in
        AgentProfile agent_profile) raises (NameInvalid);
```

Figura 33 - Sintaxe de *register_agent* ( ). (MASIF cap 3 p. 29).

Como parâmetro contém:

- *agent_name* - Nome do agente a ser adicionado à lista do *MAFFinder*.
- *agent_location* - Localização do agente.
- *agent_profile* - Perfil do agente.

A exceção pode ser:

- *NameInvalid* - A solicitação de registro do agente falhou..

Register_agent_system ( )

O método *register_agent_system ( )* registra o nome do sistema de agentes especificado na lista de sistemas do *MAFFinder*. Por se tratar de um objeto estático, quando houver necessidade de mudança de lugar é necessário de se proceda primeiro um cancelamento

do registro (*unregister_agent_system*) e posteriormente um novo registro com a nova localização. Um modelo de sintaxe é mostrado na figura 34:

```
void register_agent_system (in Name agent_system_name, in Location
        agent_system_location, in AgentSystemInfo agent_system_info) raises
        (NameInvalid);
```

Figura 34 - Sintaxe de *register_agent_system* ( ). (MASIF cap 3 p. 29).

Como parâmetro contém:

- *agent_system_name* - Nome do sistema de agentes a ser adicionado a lista de sistemas do MAFFinder.
- *agent_system_location* - Localização do sistema de agentes a ser adicionado.
- *agent_system_info* - Perfil do sistema de agentes.

A exceção pode ser:

- *NameInvalid* - Já existe um sistema de agentes registrado com este nome.

Register_place ( )

O método *register_place ( )* adiciona a localização do local espeficicado na lista de locais do *MAFFinder*. Assim como para o registro de sistemas de agente, este método não pode ser chamado mais de uma vez mesmo por que este é um objeto estático. Caso haja necessidade de mudança de local deve-se primeiro chamar o método *unregister_place ( )* e depois o *register_place ( )* para o novo local. Um modelo de sintaxe é mostrado na figura 35:

```
void register_place (in string place_name, in Location place_location) raises
        (NameInvalid);
```

Figura 35 - Sintaxe de *register_place* ( ). (MASIF cap 3 p. 30).

Como parâmetros contém:

- *place_name* - Nome do local a ser adicionado à lista.
- *place_location* - Localização do local.

A exceção pode ser:

- *NameInvalid* - Já existe um local registrado com este nome.

Unregister_agent ( )

O método *unregister_agent ( )* . Um modelo de sintaxe é mostrado na figura 36:

```
void unregister_agent (in Name agent_name) raises (EntryNotFound);
```

Figura 36 - Sintaxe de *unregister_agent* ( ). (MASIF cap 3 p. 31).

Como parâmetro contém:

- *agent_name* - Nome do agente a ser removido da lista de agentes registrados no *MAFFinder*.

A exceção pode ser:

- *EntryNotFound* - O agente não esta registrado no *MAFFinder*.

Unregister_agent_system ( )

O método *unregister_agent_system ( )* retira o nome do sistema de agentes da lista de sistemas do *MAFFinder*. Um modelo de sintaxe é mostrado na figura 37:

```
void unregister_agent_system (in Name agent_system_name) raises
        (EntryNotFound);
```

Figura 37 - Sintaxe de *unregister_agent_system* ( ). (MASIF cap 3 p. 32).

Como parâmetro contém:

- *agent__system_name* - Nome do sistema de agentes a ser retirado da lista no *MAFFinder*.

A exceção pode ser:

- *EntryNotFound* - O sistema de agentes não esta registrado no *MAFFinder*.

Unregister_place ( )

O método *unregister_place ( )* retira o local especificado da lista de locais no *MAFFinder*. Um modelo de sintaxe é mostrado na figura 38:

```
void unregister_place (in string place_name) raises (EntryNotFound);
```

Figura 38 - Sintaxe de *unregister_place* ( ). (MASIF cap 3 p. 32).

Como parâmetro contém:

- *place_name* - Nome do local a ser retirado da lista do *MAFFinder*.

  A exceção pode ser:

- *EntryNotFound* - O local especificado não esta registrado no MAFFinder.

3 - PLATAFORMAS PARA AGENTES MÓVEIS

Neste item, são apresentados cinco entre as mais representativas plataformas para agentes móveis disponíveis: Aglets/IBM, Voyager/ObjectSpace, Concordia/Mitsubishi e Grasshopper/IKV++ e MAP/Università di Catania.

Todas são baseadas em Java, o que permite que suas implementações sejam independentes das máquinas nas quais estejam instaladas.

> *"A tendência é claramente a utilização de plataformas para agentes móveis baseadas em Java, pois pode-se tirar vantagem das características de mobilidade embutidas no Java e o seu contínuo comprometimento com a tecnologia de objetos distribuídos." (Perdikeas, 1999:2004)*

Essas plataformas possuem habilidade de integração com sistemas legados através de ORBs (exceto Aglets), algumas vezes proprietários.

Apesar da diversidade de características que cada uma possui, todas possuem algumas características em comum, além da utilização de Java. Possuem sua própria biblioteca Java, a partir do qual podem ser criados agentes através da instanciação de classes.

3.1 - Aglets / IBM

Após o surgimento da linguagem Java em 1995 o laboratório de pesquisas da IBM, em Tóquio/Japão, decidiu desenvolver um projeto, denominado Aglets, que é um ambiente para programação de agentes móveis desenvolvido na linguagem Java. O Aglets se utiliza do Java, da Internet e das vantagens de agentes móveis.

A primeira versão foi em criada em 1996, no laboratório de pesquisas da IBM em Tóquio/Japão. A comunidade acadêmica tinha acesso a uma versão fornecida pela IBM (http://www.trl.ibm.com.jp), que posteriormente liberou o programa com seu código fonte, que pode ser encontrado no Sourceforge.net (http://www.sourceforge.net/aglets).

A primeira versão foi disponibilizada ainda em 1996 através de um *kit*, denominado ASDK, acessível gratuitamente pela Internet. Posteriormente outras versões foram sendo liberadas. A versão beta 3 (1.0.3) esteve em uso até 25 dezembro de 2000, quando foi substituída pela 1.1.0, a qual possui seu código fonte liberado pela IBM e publicado na *Internet*, compatível com JDK versão 1.1.x.

O ASDK possui uma interface gráfica de usuário (GUI) denominada Tahiti, que pode ser entendida como a representação gráfica de um contexto, através da qual pode-se criar, despachar, monitorar, extinguir agentes. Pode também estabelecer privilégios de acesso do agente ao servidor. Possui ainda recursos de persistência, clonagem, identificação única de agentes, envio de mensagens síncronas ou assíncronas, multicast de mensagens, tunelamento http e modelo de eventos.

Para sua utilização é necessário que, na origem e no destino, tenham sido instalados uma máquina virtual Java e um servidor Aglets, além de, pelo menos, um aglet.

3.1.1 - Elementos básicos

Aglet - Um aglet é um objeto Java e também um agente móvel, ou seja, tem capacidade de mover-se de um ponto a outro da rede, de um modo autônomo.

Proxy - O Proxy atua como um representante de um aglet protegendo-o de um acesso direto por parte de outros, protegendo seus métodos públicos e escondendo sua real localização.

Contexto - É um objeto estacionário que promove um ambiente de execução para o gerenciamento e manutenção dos agentes após serem criados ou em execução. Um endereço de rede pode conter vários servidores, e um determinado servidor pode manter vários contextos.

Mensagem - É um objeto para a troca de mensagens entre os aglets, que pode ser feito em modo síncrono ou assíncrono.

Resposta futura - Utilizada quando do envio de mensagens assíncronas para posterior envio e recebimento.

Identificador - Cada aglet deve receber uma identificação única que permita sua identificação onde quer que o mesmo se encontre, e deve acompanhá-lo por toda a sua existência.

3.1.2 - O modelo de eventos

A programação do Aglets é baseada em eventos. Três tipos de *Listeners* (ouvintes) capturam determinados eventos em um aglet e ativam determinadas ações.

Clone listener - é utilizado para adoção de ações específicas quando da clonagem de um aglet. Pode atuar imediatamente antes, ou logo a seguir da clonagem ocorrer.

Mobility listener - é utilizado quando da movimentação, seja um envio ou retração, de um aglet. Ativa ações específicas a serem tomadas quando o agente está para ser enviado a outro contexto, quando esta para ser recuperado no contexto, ou quando chega a um novo contexto.

Persistence listener - permite a adoção de ações quando se solicita a exclusão de um agente. Tais como permitir o encerramento prévio de atividades do agente.

3.1.3 - Operações fundamentais

Podem ser divididas em quatro categorias:

Criação / Clonagem - É a primeira etapa na utilização de um aglet. Para isso o mesmo deve ser instanciado de uma classe aglet ou clonado de um aglet existente. A instanciação do agente se dá através da obtenção de um contexto e da chamada do método para criação do aglet. Exemplo de linha de programa criando um aglet:

getAgletContext ().createAglet (getCodeBase (), "Some Aglet", null);

Inicialização / Suspensão - Controlam a persistência de um aglet. Estes eventos são denominados também de ativação e desativação.

A desativação permite a colocação do agente em um estado de espera. Apesar de ainda se manter no contexto, o mesmo libera as linhas de execução que criou e passa para um armazenamento secundário. A ativação executa o processo inverso.

Quando da solicitação da desativação um agente (aglet) pelo método *deactivate ()*, um outro método denominado *onDeactivating ( )*, ativado por um *persistence listener*, informa ao aglet que o mesmo será desativado, permitindo que o mesmo encerre suas atividades.

Envio / Retração - Relativos à movimentação dos aglets. O primeiro modo de envio de um aglet a outro local é através do método *dispatch ()*.

Após a chegada do aglet ao destino, o método *onArrival ( )* é executado e na seqüência o *método run ()*. Exemplo de utilização do método *dispatch ()*:

Public final void Aglet.dispatch (URL destination)

O segundo modo é através do método *retractAglet ()*, no qual o sistema de origem chama o agente de volta. Este método somente pode ser utilizado após a utilização do método *dispatch ()* pelo menos uma vez.

De modo análogo ao processo de desativação, a retração provoca a utilização de outros métodos, acionados através de *mobility listeners*, para a transferência do aglet de volta

ao local de origem. O primeiro deste métodos é o *onReverting ( )*, que informa ao aglet que o mesmo está para ser transferido, permitindo-lhe que encerre suas atividades. O outro método é o método *onArrival ( )* que é executado já no local de origem após a transferência de volta do aglet e antes de sua reativação.

Um exemplo de utilização do método *retractAglet ( )*:

AgletContext.retractAglet (URL contextAdress, AgletID aid)

Destruição - Quando da destruição de um aglet, todas as referências entre o contexto e o aglet são eliminadas. Todas as linhas de execução relacionadas a este aglet serão devolvidas ao sistema, e a memória utilizada será colocada à disposição para a próxima coleta de memória (*garbage collector*). Embora não seja o usual, pode-se especificar uma coleta imediata, se necessária.

Similar ao processo de desativação, o sistema Aglet possui um recurso que permite ao agente (aglet) ser avisado quando da solicitação de sua destruição. Ao ser chamado o método *Aglet.dispose ( )*, o sistema chama o método *Aglet.onDisposing ( )*, através de uma listener associada a este evento, permitindo então que o mesmo encerre suas atividades e se prepare para sua extinção.

Exemplo de chamada do método para destruição do aglet:

Public final void Aglet.dispose ( )

3.1.4 - Padrões de projeto

Uma importante parte do modelo de programação Aglet são os padrões de projeto para agentes. Estes padrões são como modelos pré-estabelecidos para utilização com agentes permitindo uma importante reciclagem do código e o aproveitamento de soluções a problemas mais comuns.

Estes padrões são utilizados no JAVA, e sua idéia foi aproveitada no Aglets.

Estes padrões podem ser classificados em:

Traveling Patterns - Relativos à movimentação dos aglets, e utilizados no gerenciamento da mobilidade, tais como roteamento e qualidade de serviço. Um exemplo deste tipo seria o padrão de itinerário, que pode manter listas com o roteamento para múltiplos destinos e definir o que deve ser feito no caso de um destinatário não existir em um objeto. Esta abordagem de itinerário com uso de objetos permite uma reutilização e atualização constante dos destinos.

Task Patterns - Relativos à conclusão das tarefas submetidas aos aglets, executam tarefas descritas, subdividindo-as para a sua resolução por um ou mais aglets, de uma maneira paralela e ou colaborativa. Um modelo deste padrão é o mestre-escravo (master-slave).

Neste modelo, um aglet (mestre) após ser instanciado e inicializado cria um ou mais aglets (escravos) que irão executar determinada tarefa retornando o resultado ao aglet-mestre para análise e/ou consolidação de resultados. Uma implementação do modelo mestre-escravo em Java foi incluída no Aglets.

Interaction Patterns - Relativo à comunicação entre os aglets, permite a cooperação entre os mesmos. Neste padrão podem ser citados os modelos *Meeting* e *Messenger*.

O padrão *Meeting* provê uma maneira de interação entre dois ou mais aglets em um certo *host*. Os aglets podem se despachar para um determinado ponto de encontro (*meeting place*) e interagir. Já o padrão *Messenger* provê uma comunicação entre aglets, usando outro aglets como veículo destas mensagens. A implementação deste dois padrões, em Java, foi incluída no Aglets.

3.1.5 - O pacote Aglet

É composto por três classes abstratas e três interfaces, a partir das quais toda a implementação do agentes aglet pode ser feita.

A classe Aglet - *com.ibm.aglet.Aglet* - É a classe base para o desenvolvimento de aglets. Define métodos para controle do ciclo de vida do agente.

A interface AgletProxy - *com.ibm.aglet.AgletProxy* - Age como assessor do aglet prevenindo acesso direto ao aglet. As solicitações destinadas ao aglet são interceptadas pelo AgletProxy que consulta o gerenciamento de segurança visando determinar se o acesso é ou não permitido. Pode inclusive estar em um contexto diferente do aglet.

A interface AgletContext - *com.ibm.aglet.AgletContext* - Equivalente ao conceito de local, provê um ambiente aos agentes presentes em um determinado contexto. Um aglet passa a maior parte de sua existência em um ou mais contextos. É no contexto que o aglet é criado, suspenso, destruído. É através desta interface que o agente obtém as informações sobre o contexto em que se encontra, podendo, inclusive, criar outros aglets, por exemplo.

A classe Message - *com.ibm.aglet.Message* - A comunicação entre aglets se dá por meio de objetos desta classe. O conteúdo da mensagem é passado como um parâmetro do método *handleMessage ( )*.

A interface FutureReply - *com.ibm.aglet.FutureReply* - O objeto definido nesta interface é retornado pelo método de envio de mensagem assíncrono e é usado para manipulação do resultado recebido posteriormente. Através desta interface o receptor pode determinar se uma resposta já está disponível ou esperar até que um tempo limite ocorra e continue sua tarefa.

A classe AgletID - *com.ibm.aglet.AgletID* - Esta classe abstrai a identificação dos aglets. Cada aglet deve possuir uma identificação única e que o acompanhe por toda a sua existência.

3.1.6 - Transmissão do agente

O procedimento de transporte do agente se inicia com a chamada do método *dispatch ( )* em um Aglet. Após esta chamada outro método, *onDispatching ( )*, é chamado. Este método é uma das particularidades do Aglet, é a preparação do agente para transferência.

O método *onDispatching ( )* informa ao agente que sua transferência se iniciará dando oportunidade ao mesmo de encerrar atividade em andamento ou mesmo recusar-se a ser transferido. Em se iniciando a transferência, e quando o servidor de destino detecta o Aglet, o mesmo chama o método *onArrival ( )* para recepção do agente e a seguir o método *run* para iniciá-lo. O servidor de destino informa então à origem o novo *proxy* do agente. As comunicações entre a origem e o agente em seu novo destino passam a ser feitas por este *proxy*. Por exemplo, para como chamá-lo de volta (Retract).

O servidor do destino monitora uma porta determinada, em geral a porta 434 (Estabelecida pela RFC1700 como para agentes móveis). Os agentes são atendidos por ordem de chegada.

O protocolo utilizado pelo Aglets, denominado ATP (*Agent Transfer Protocol*), é um protocolo para uso em sistemas distribuídos baseados em agentes, podendo ser utilizado para transferência de agentes móveis através de uma rede.

Projetado visando a *Internet*, é um protocolo simples e independente de plataforma. Cobre áreas tais como: serviço de nomeação de agentes, identificadores de agentes, transporte de agentes. O EBNF (Extended Backus-Naum Form) deste protocolo pode ser encontrado no anexo 1.

O protocolo define quatro métodos padrão para serviços com agentes, *Dispatch*, *Retract*, *Fetch* e *Message*:

- *Dispatch* - para se enviar um agente de A para B, o A envia um *dispatch* para B. O agente segue no corpo da requisição. Após receber o agente, B envia uma resposta a A.
- *Retract* - para chamar o agente de volta, A envia um *retract* para B. Após receber a solicitação, B envia uma resposta com o agente incluído.
- *Fetch* - para executar um agente enviado por A, o servidor B pode necessitar de alguma classe que não possua. Então B envia um fetch_class solicitando à classe necessária a execução do agente.
- *Message* - para envio de uma mensagem de um agente em A para um agente em B, A envia uma *message* para B. B acusa o recebimento.

Apesar da utilização de uma API de comunicação derivada do MASIF para comunicação interna entre a camada de execução e a de comunicação, esta interface não está disponível como uma interface externa. Para isso, o Aglets suporta o protocolo proprietário ATP e o Java RMI.

Uma camada de transporte baseada no CORBA IIOP deve ser fornecida, de acordo com a IBM, em versões futuras do Aglets. Todavia como o Aglets tornou-se um programa de código aberto, esta possibilidade é pequena.

3.1.7 - Segurança

É permitida a adoção de políticas de segurança, que podem ser configuradas através da interface gráfica de usuário (GUI).

Pode-se proteger o host de agentes maliciosos, através da autenticação de domínios. Podem ser autenticados o usuário, o host (servidor) e o agente.

A integridade dos agentes durante a transmissão pode ser feita utilizando-se criptografia de chave simétrica..

3.2 - Voyager / ObjectSpace

O Voyager foi elaborado pela ObjectSpace, uma empresa fundada em 1992 e baseada em Dallas/EUA.

O Voyager é uma plataforma para o desenvolvimento de aplicações distribuídas, baseada em Java com recursos para lidar com agentes móveis. Possui capacidade para criar e mover objetos remotamente, através da utilização de mensagens, podendo mover estes objetos entre programas.

É composto de um ORB (Object Request Broker) com suporte para utilização de objetos e agentes móveis.

Foi desenvolvida em Java aliada, a utilização de ORBs, o que lhe confere a vantagem de poder lidar com sistemas legados e ao mesmo tempo utilizar a tecnologia de agentes móveis. Tira proveito dos benefícios desta nova tecnologia e integrando-a com às já existentes.

> *Um dos principais destaques da linguagem Java é a habilidade de adicionar classes em uma máquina virtual em tempo de execução. Esta capacidade possibilita a utilização de objetos móveis e agentes autônomos como uma outra ferramenta para a construção de sistemas distribuídos. A adição desta capacidade a antigas tecnologias distribuídas é quase impraticável e resulta em infra-estruturas difíceis de usar.* (Glass In: Milojicic D., 1998:612)

Possui como clientes, dentre outros, a Galileo International e a General Motors/OnStar. A Galileo International é uma empresa que opera um sistema de distribuição eletrônico de alcance mundial, unindo mais de 500 companhias aéreas, e dezenas de aluguel de carros, hotéis e outras empresas relacionadas a agências de turismo, clientes corporativos e a consumidores, via portais na *Internet*. Processou no ano de 2000, um terço das mais de um bilhão de reservas / emissão de bilhetes aéreos no mundo. É remunerada pela quantidade de operações efetuadas, e sendo esta a sua principal fonte de receita, é interesse da empresa permitir a utilização de seus sistemas pelo maior número possível de clientes. Todos estes clientes podem acessar sua base de dados, por meio de vários tipos de acessos diferentes, tais como: sistemas de informática proprietários, telefones celulares, *pagers*, assistentes pessoais digitais (PDAs) e *Internet*. Ao utilizar um destes meios de acesso, o cliente efetua uma consulta, por exemplo dos vôos entre duas cidades em uma determinada data, desejando obter horários, tarifas e outras informações.

Ao acessar o sistema, via quaisquer meios citados, o cliente digita os dados que dispõe: origem, destino e data. A partir de então estes dados são enviados às companhias aéreas em agentes móveis. Os sistemas no destino ao receberem estes agentes, lêem o seu

conteúdo e processam as informações, devolvendo o agente a sua origem. Os agentes retornam então à origem carregando os dados pesquisados. Os dados são então classificados como solicitado pelo cliente, em ordem crescente de tarifas ou de tempo total de vôo. O cliente pode então efetuar sua reserva, comprar os bilhetes ou ainda incluir outros itens como aluguel de veículo e reserva de hotel.

A OnStar, uma divisão da General Motors, é a empresa responsável pelo desenvolvimento e manutenção do sistema de comunicação sem fio, disponível para vários modelos GM, e pela central de atendimento da OnStar. Tal sistema fornece informações em tempo real sobre serviços disponíveis no local em que o veículo se encontra, por exemplo: hotéis, restaurantes, postos de gasolina e centros de manutenção. Além de abertura remota do veículo, localização de veículo roubado. A falha de determinados componentes do veículo também pode ser repassada ao centro de manutenção que já pode então preparar-se para o serviço a ser executado.

Estes dois exemplos utilizam sistemas de informática nos quais o agente móvel ao chegar ao destino freqüentemente tem de acessar informações em sistemas legados, via ORBs, e depois retornar as informações, via protocolos como HTTP e WAP.

A proposta do Voyager é então adicionar recursos disponíveis nos sistemas baseados em agentes móveis e Java ao sistemas já em operação, preservando os investimentos já feitos e os futuros. Isso possibilita a integração em tempo real de grandes sistemas de código estruturado com aplicações baseadas em Web, orientadas a objeto.

3.2.1 - Elementos básicos

A versão utilizada neste trabalho é a ObjectSpace Voyager 4.0 International Trial Version noSSL Encryption, tornada disponível em 31/10/2000, com algumas de suas funções desabilitadas e somente disponíveis na versão profissional.

Voyager ORB - ORB com vários pacotes para utilização de objetos móveis e agentes autônomos. O pacote Voyager foi organizado em diretórios do seguinte modo:

- voyager/bin - Contém os utilitários do Voyager. Tais como o servidor voyager (voyager); o gerador de arquivos de interface (igen); a conversão entre Java e IDL (cgen); gerador de interface Java usado pela XML (*Extensible Markup Language*); utilitário de instalação (aclinst); dentre outros.

- voyager/lib - Contém os arquivos com as classes Voyager e de terceiros. Na instalação padrão é necessário o arquivo voyager.jar e os arquivos de terceiros que se fizerem necessários. Na instalação em dispositivos móveis são necessários o lightclient.jar, lightjndiclient.jar e jndi.jar. Outros arquivos .jar são fornecidos.
- voyager/idl - Contém os arquivos .idl para suporte ao CORBA.
- voyager/examples - Contém vários exemplos para fins didáticos.
- voyager/doc - Documentação do pacote.

A plataforma Voyager é compatível com JDK 1.1 e 1.2 e possui três módulos adicionais que podem ser utilizados em conjunto com Voyager ORB, que são: *Security*, *Transactions*, e *Application Server*.

Voyager Security - Provê serviços de segurança tais como: autenticação, autorização, privacidade e integridade dos dados. Implementa os protocolos socks 4 e 5, e tunelamento em HTTP para comunicações seguras.

Voyager Transactions - Fornece serviços de atomicidade, consistência e isolação dos dados. Adapta também conexões JDBC ao *Transaction Service* do Voyager.

Voyager Application Server - Implementação para o modelo EJB (Enterprise JavaBeans). O EJB é uma arquitetura para aplicações distribuídas, seguras, baseadas em componentes para o Java. Possui ferramentas para automatizar a criação de componentes EJB.

3.2.2 - Operações fundamentais

Migração - O Voyager tem a habilidade de mover qualquer objeto Java presente em uma plataforma para outra. Entretanto, somente objetos instanciados da classe *agent* podem, após mover-se, continuar sua execução no destino.

A capacidade de mover outros objetos além dos agentes confere uma funcionalidade adicional de mover objetos estáticos, o que pode ser útil quando se deseja movimentar um objeto, beneficiando-se da localidade, sem a necessidade de conferir-lhe autonomia.

Persistência - É definida uma interface *IVoyagerDb* que pode armazenar um objeto Voyager em um banco da dados de uma maneira relacional ou orientada a objeto.

Segurança - Impõe, através de métodos, restrições a objetos originários de outras redes tais como: impedimento ao monitoramento de uma determinada porta ou mesmo a criação de alguma janela.

Ciclo de vida - O Término da vida de um agente pode ser definido para se dar quando da ocorrência de um determinado fato ou então por tempo determinado. No caso de tempo determinado, pode ser especificado um tempo de vida, uma determinada hora para finalização, ou estabelecido que o agente viverá permanentemente. Estes valores podem ser alterados em qualquer momento.

Mensagens e Eventos - Podem ser enviadas mensagens síncronas, *one-way* e de uso futuro (*Future Messages*). As one-way retornam imediatamente sem retornar valores a não ser exceções. As *Future Messages*, retornam imediatamente e podem ser consultadas no futuro. Estas mensagens podem ser construídas tanto em tempo de execução quanto de compilação.

3.2.3 - O pacote Voyager

As principais classes constituintes do Voyager, relativos a atividades de gerenciamento, nomeação, mobilidade e notificação de eventos, são:

- *com.objectspace.voyager* - Contém as classes principais para o funcionamento do Voyager. Dentre elas: *Namespace* que contêm métodos para associar um nome a um objeto; e *Factory* que contém métodos para criação remota de objetos.

- *com.objectspace.voyager.mobility* - Define classes que permitam a movimentação de objetos serializáveis entre as ORBs.

- *com.objectspace.voyager.agent* - Provê extensões, para autonomia e chamada de volta de objetos, ao pacote *mobility*. Os agentes são criados por instanciações desta classe.

- *com.objectspace.lib.timer* - Provê interfaces e classes para *listeners* relativos a controle de tempo e alarmes.

- *com.objectspace.voyager.corba* - Contém a implementação CORBA do Voyager.

- *com.objectspace.voyager.corba.naming* - Contém a implementação para o serviço de nomeação CORBA para o Voyager.

- *com.objectspace.voyager.loader* - Contém mecanismos para o carregamento de classes e proxies do Voyager.

- *com.objectspace.voyager.system* - Provê interface para serviços que serão adicionados ao servidor Voyager.

- *com.objectspace.voyager.message* - Provê recursos para lidar com mensagens, tais como interface e *listeners* para eventos.

- *com.objectspace.voyager.space* - Provê interfaces e classes que possibilitam o recurso de criação de grupos para notificação de eventos. Possui ainda outros dois pacotes para notificação de eventos: *multicasting* e *publishing*.

- *com.objectspace.voyager.transport* - Provê interfaces para permitir o reconhecimento e processamento de aplicações a nível de protocolo tais como IIOP e HTTP; e protocolos de transporte como UDP e IPX.

- *com.objectspace.voyager.management* - Contém interfaces e classes para configuração e gerenciamento.

- *com.objectspace.voyager.tcp.socket* - Contém interfaces para implementação de *sockets* do lado do servidor e do cliente utilizados pelos pacote *management*.

- *com.objectspace.voyager.rmi* - Contém classes relativas à implementação RMI pelo Voyager.

3.2.4 - Transmissão do agente

Quando da iniciação de um programa Voyager, automaticamente são inicializados serviços para tráfego de rede, controle de tempo, coleta de lixo. Para cada programa Voyager é atribuída uma porta, que pode ser especificada pelo usuário ou ser escolhida aleatoriamente. Exemplo de inicialização de um programa voyager por um objeto:

Voyager.startup(7000); // aqui foi especificada a porta 7000

Para mover-se de um programa a outro, normalmente com a intenção de obter as vantagens de localidade, um agente chama a mensagem *moveTo ( )* em si mesmo. São incluídos, no comando, o destino do agente e o método a ser chamado quando no destino. Exemplo: O agente é enviado ao servidor tokyo na porta 9000, sendo o método *atTokyo* chamado quando da chegada ao destino. Exemplo:

```
Public void travel ()
  {
  MoveTo ("tokyo:9000","atTokyo");
  }
```

```
Public void atTokyo ()
  {
```

```
// código a ser executado quando em tokyo:9000
}
```

3.2.5 - Segurança

O Voyager tem sua segurança implementada pelo módulo Security, que impõe restrições aos objetos ou agentes, além daquelas já implementadas pelo Java. Possui, como mecanismos de segurança, a utilização de Socks 4 e 5, e tunelamento HTTP. Não são incluídas no Voyager mecanismos de criptografia ou autenticação de *hosts* ou agentes.

3.3 - Concordia / Mitsubishi Electric

O Concordia é uma plataforma para desenvolvimento e gerenciamento de agentes móveis, escrita em Java e extensível a qualquer equipamento compatível com esta linguagem.

Foi desenvolvido pela Mitsubishi Electric Information Technology Center America (MEITCA), com sede em Massachusetts/EUA.

A versão utilizada neste trabalho é a 1.1.7 lançada em 1° de janeiro de 2001 compatível com Java 1.1.7B. Podendo ser utilizada com versões Java não anteriores a 1.1.3.

Tem por finalidade e objetivos, através da transferência de agentes, e do uso de mobilidade, suporte a colaboração, persistência, transmissão confiável e segurança do agente permitir: utilização de hardwares portáteis que possuam recursos muito limitados (*laptops*, PDAs, etc.); utilização de bancos de dados remotos através de utilização de conexões temporárias ou precárias; melhor aproveitamento das conexões de rede, sejam elas temporárias ou não; crescimento das redes atuais sem aumentar sua complexidade; e permitir atualizações de programas sem interrupção do funcionamento da rede.

Como exemplos podem ser citados: possibilitar acesso a sistemas legados por parte de usuários móveis; possibilitar aplicativos a receberem um solicitação do usuário mesmo que este não tenha uma interface de usuário ativa e que possa receber os resultados quando completados; permitir a usuários móveis que acessam a rede através de meios problemáticos a processarem os dados localmente no servidor e receberem somente os resultados selecionados; possibilitar o monitoramento de rede com um mínimo de *overhead*, pois pode ser feito sem o envio/recebimento sistemático de dados; permitir a utilização de múltiplas plataformas em uma rede que necessite coletar e distribuir dados por meio de uma combinação de laptops, PDAs, smartphones, email, voicemail, pagers e fax.

3.3.1 - Partes principais

O Concordia é constituído de três partes: uma máquina virtual Java, um servidor Concordia (Fig.39) e de pelo menos um agente. Toda a codificação dos componentes é em Java.

A máquina virtual Java, denominada JVM (*Java Virtual Machine*), em muitos casos já está instalada e em uso por outros aplicativos.

O agente móvel é um objeto Java.

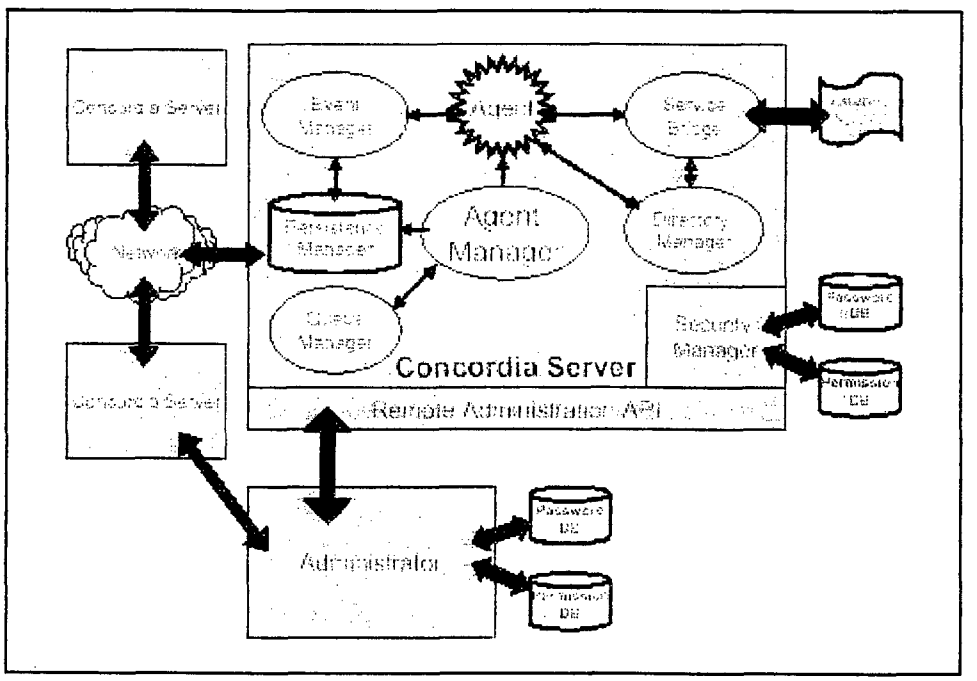


Figura 39 - Arquitetura Concordia.

3.3.2 - Componentes do Servidor

O servidor Concordia é composto de oito módulos que, instalados e em execução, lidam com todas as tarefas relacionadas ao manuseio de agentes:

a) Gerenciador de agentes - *Agent Manager* - Provê a infra-estrutura de comunicação que permite ao agente, um objeto Java, a mover-se pela rede. Provê o ambiente para criação, execução e destruição do agente. Gerencia o ciclo de vida do agente. Possui como característica o fato de abstrair a interface de rede, não sendo necessário ao programador do agente incluir código referente a especifidades da rede.

b) Gerenciador de segurança - *Security Manager* - Tem como função manter a segurança e a integridade dos dos agentes móveis e dos recursos da máquina na qual está instalado o servidor Concordia. É responsável pela identificação dos usuários, autenticação dos agentes, proteção dos recursos do servidor, segurança e integridade do agente e dos dados que carrega, autoriza ainda o carregamento dinâmico das classes Java necessárias ao funcionamento do agente. Possui uma interface de usuário para monitoramento e configuração de atributos de segurança relativo aos usuários e serviços a serem prestados.

O estabelecimento da segurança pode se dar de vários modos: para sistemas altamente seguros pode-se não estabelecer nenhum tipo de credencial de acesso; para redes que utilizam canais públicos (por exemplo, *Internet*) pode-se utilizar criptografia; Para garantia de identificação do usuário e segurança dos dados pode-se utilizar de mecanismos mais fortes para criptografia e autenticação.

c) Gerenciador de persistência - *Persistence Manager* - Mantém um registro do estado dos agentes móveis em trânsito pela rede, caso ocorra algum problema com a rede ou servidor, evitando a necessidade de uma re-inicialização do agente, podendo o mesmo recomeçar do ponto anterior em que se situava no momento da falha.

d) Gerenciador de comunicação entre agentes - *Inter-Agent Communication Manager* - Controla o registro de eventos e mensagens de e para os agentes móveis, podendo alcançá-los em qualquer ponto que estejam e a qualquer momento. Conjuntamente com o servidor Concordia, pode distribuir eventos, se necessário, para sincronização e compartilhamento de dados entre os agentes ou *multicasting*.

e) Gerenciador de fila - *Queue Manager* - Responsável pela garantia de entrega dos agentes móveis entre servidores Concordia, especialmente útil em redes não-confiáveis.

Mantém os agentes enquanto aguardam a oportunidade para execução de suas tarefas, de acordo com prioridades que pode estabelecer para acesso a determinado nó da rede, mantendo ainda seus estados. Refaz solicitações quando o servidor Concordia for desconectado da rede por algum motivo.

f) Gerenciador de diretório - *Directory Manager* - Provê o serviço de nomeação para agentes e aplicações, através do qual os agentes localizam os serviços na rede. Pode ser configurado de várias maneiras, de acordo com a necessidade dos serviços a serem prestados.

g) Gerenciador da administração - *Administration Manager* - Permite a administração remota do Concordia, inclusive por um único administrador, caso necessário.

Este módulo permite o gerenciamento simultâneo de muitos servidores Concordia através de uma interface de usuário.

O módulo de administração do Concordia é responsável pelo gerenciamento de todos os serviços, tais como: gerenciamento de agentes (*Agent Manager*); gerenciamento da segurança (*Security Manager*); gerenciadores de eventos (*Event Managers*)

h) Biblioteca de ferramentas para agentes - *Agent Tool Library* - Conjunto de APIs (*Application Program Interface*) do Concordia para administração, transporte de agentes e alterações em aplicativos associados denominados. Inclui todas as classes necessárias ao desenvolvimento de agentes móveis, inclusive, a classe *Agent* e outras derivadas de classes do Java, com interfaces para o Concordia.

3.3.3 - Outros componentes do pacote Concordia

O Concordia possui quatro pacotes principais, todos eles com contém interfaces, classes e exceções:

O *com.meitca.Concordia*, contém classes para lidar com agentes, sua transferência e controle de itinerários, além de interfaces para lidar com eventos relativos a agentes e sua transferência.

Serviços para Aplicações Específicas (*Aplication Specific Services*) - *com.meitca.Concordia.service* - confere ao servidor Concordia a habilidade de conectar agentes a aplicativos específicos por meio de uma *servide bridge*. Uma *service bridge* é um objeto Java instalado em um servidor Concordia e quando inicializado junto com o servidor é automaticamente registrado junto ao serviço de nomeação do servidor, passando então a possuir um nome a ele associado. Permite assim a ligação entre agentes que chegam ao servidor Concordia e um aplicativo específico ligado a esta máquina, por exemplo um banco de dados. Possui API's para facilitar, aos desenvolvedores, a criação de serviços.

Colaboração (*Collaboration*) - o pacote *com.meitca.Concordia.collaborate* - Permite que vários agentes trabalhem em conjunto ou coordenem sua ações, formando grupos, e possui API's para colaboração entre agentes tanto no modo síncrono quanto assíncrono. Permite também uma sincronização entre agentes sem troca de dados.

Eventos distribuídos *(Distributed Events)* - *com.meitca.Concordia.event* - Exerce a atividade relativa a ocorrência de eventos.

Em associação com os gerenciadores de eventos podem ser executadas atividades tais como: filtragem de eventos; escolha de notificação de mensagem síncrona ou assíncrona;

controle de fila para aventos destinados a determinado agente; encaminhamento de eventos a agentes em trânsito; re-inicialização do gerenciador de eventos após a ocorrência de alguma falha; mecanismo de registro de eventos.

3.3.4 - Transmissão do agente

A transmissão do agente é feita pelo gerenciador de agentes, também denominado em alguns documentos como: servidor *Conduit*. Baseia-se no Java RMI para a transferência de agentes, o qual envia somente uma imagem do agente. Todas as classes necessárias para a execução do agente são enviadas na medida em que se tornem necessárias.

Quando um agente necessita ser transferido, o mesmo invoca o método local do servidor *Conduit*. A execução do agente é suspensa, e uma imagem persistente do estado do agente é criada. O servidor *Conduit* checa então o itinerário constante no agente visando estabelecer o destino. Após uma imagem do agente é enviada ao servidor *Conduit* no Concordia da máquina de destino. O agente é então armazenado no destino. Somente após o envio de confirmação de recebimento pelo destino, é que a origem procede a eliminação da cópia do mesmo na origem.

Convém ressaltar que, no Concordia, a migração é considerada do tipo fraca, ou seja, somente o estado do agente é serializado e não a pilha e contadores dos *threads* em execução pelo agente no momento da transferência.

O agente não é questionado se aceita ou não ser movido, assim como não recebe aviso prévio de que será transportado. Isto se torna um benefício com relação à agilidade de transferência, todavia impede que o agente se prepare mais adequadamente para o transporte. Por exemplo, finalizando alguma atividade em andamento.

A garantia de transmissão correta do agente é dada pelo *Queue Manager* que, em conjunto com o servidor *Conduit*, executam as atividades de conexão com outros *Queue Managers*.

Não há necessidade de instalação do Concordia na máquina cliente caso esta somente vá executar tarefas de recebimento, envio e execução de agentes. Uma classe abstrata denominada *ConcordiaApplet*, extendida da classe *Applet* do Java, implementa uma classe do Concordia denominada *AgentTransporter*.

Para isso pode-se utilizar o objeto *AgentTransporter* o qual pode ser utilizado diretamente pela máquina virtual Java. Este *AgentTransporter* age como um servidor

Concordia mais simples que poderia ficar instalado em um navegador como um *plug-in*, por exemplo.

A aplicação pode também implementar uma listener para detecção de chegada de agentes, obtendo então um referência a eles por meio do *AgentTransporter* para poder, então, invocar os métodos do agente

Em um teste efetuado por Samaras, et alli, ocorreram problemas de violação pois o *AgentTransporter* tentava acessar diretamente recursos do sistema. Para livrarem-se deste problema instalaram manualmente várias classes na máquina cliente.

A comunicação de eventos de e para os agentes se dá por um quadro de avisos controlado pelo gerenciador de eventos (*Inter-Agent Communication Manager*) o qual rastreia os movimentos dos agentes. Os objetos móveis (agentes) que necessitam receber ou postar determinados eventos devem registrar-se com o gerenciador de eventos de um ou mais servidores Concordia.

Para interagir com outros aplicativos, os agentes utilizam-se de métodos em uma *Service Bridge*. Esta *Service Bridge* age como uma interface entre o Concordia e os recursos ou aplicativos instalados na máquina. Este serviço elimina a necessidade dos usuários de permanecerem conectados o tempo todo com o aplicativo. Neste caso, o agente envolvido atua com o aplicativo via o *Service Bridge* sem necessidade de uma interface com o usuário aberto na tela. A conexão pode ser restabelecida pelo agente posteriormente para envio dos resultados, ou o mesmo pode utilizar-se de outras vias para entrega dos resultados da consulta.

3.3.5 - Protocolo

O Concordia utiliza-se do método de invocação remota (*Remote Method Invocation - RMI*) do Java para buscar métodos em objetos em execução em outras JVM. Por estar contida no JDK, o RMI está disponível onde haja uma JVM em funcionamento.

Uma das vantagens do RMI é a possibilidade de baixar o *bytecode* de uma classe de objeto, caso a classe não esteja na máquina de destino.

À medida que seja necessário, o método presente em outra JVM pode ser chamado. Um programa ou objeto Java pode conseguir uma referência ao método desejado, através de uma consulta a lista de métodos disponíveis ou recebendo uma referência passada como um argumento ou retorno de função.

A iniciação de um agente pode ser feita através da criação de uma classe que implementa um agente de interface, da localização de um servidor e solicitando a aceite do objeto (agente) pelo servidor.

A implementação do agente pode ser transferida ao servidor e nele executada. O método para aceitação do agente iniciaria uma nova linha de execução para o agente, invocaria o método *run ( )* e retornaria. O agente continuaria sua execução após o retorno daquele método.

O agente poderia migrar para um outro servidor através da invocação do método de aceitação no servidor de destino. Passando então ele mesmo como o agente e ser recebido e removendo o agente original do servidor de origem.

3.3.6 - Segurança

São oferecidas três opções: a utilização do protocolo de segurança SSL3 para transmissão do agente de um sistema Concordia a outro; a utilização de outro protocolo estabelecido pelo usuário; ou mesmo nenhum protocolo de segurança.

3.4 - Grasshopper / IKV++

O Grasshopper é uma plataforma para o desenvolvimento e utilização de agentes móveis, escrita em Java. Foi desenvolvido pela IKV++/Alemanha, e teve sua primeira versão lançada em agosto de 1998.

Tem por objetivo, permitir a elaboração e utilização de agentes móveis. Proporciona ferramentas para o desenvolvimento e gerenciamento de agentes, ou seja, criação, envio a outro *host*, recebimento, inicialização, suspensão, cópia, extinção, localização do agente, envio de mensagens, etc.

Proporciona ainda suporte ao CORBA e compatibilidade com o OMG MASIF (*Mobile Agent System Interoperability Facility*). De acordo com o fabricante, todas as versões são compatíveis com o MASIF, todavia da versão 2.1 em diante este recurso foi colocado em separado com um *add-on*. Possui ainda outro add-on para compatibilidade com a FIPA.

É compatível com JDK 1.2 e 1.3, possui versões para os sistemas Windows, Unix e uma versão compacta para uso em computadores portáteis, desde que os mesmos tenham capacidade para Java. Vem sendo utilizado por companhias tais como: Alcatel/Bélgica, Ericsson/Suécia, Hitachi/Japão, Matra-Nortel/França e Nortel/Reino Unido, além de centros de

pesquisa de Universidades na América do Norte (EUA e Canadá) e União Européia (Reino Unido, França, Itália, Bélgica, Alemanha, Portugal).

3.4.1 - Partes Principais

O Grasshopper é constituído basicamente de um sistema de agentes (agência), agentes, componente de registro. (Fig.40)

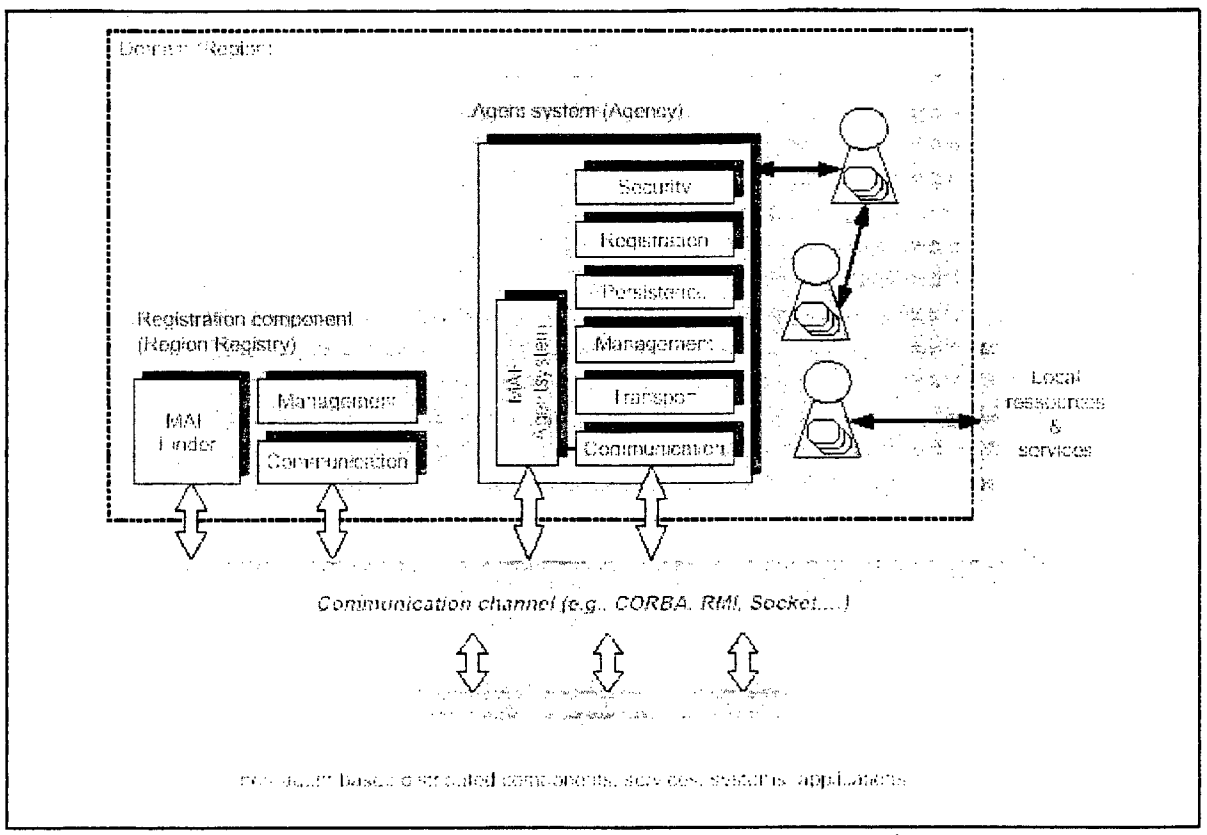


Figura 40 - Arquitetura Grasshopper (Grasshopper - A platform for mobile software agents)

O sistema de agentes é um processo Java que permite o gerenciamento das atividades dos agentes presentes.

O agente é um objeto java.

O componente de registro mantém informações sobre as agências e todos os agentes presentes nestas agências.

Região é um domínio no qual estão situadas as agências, com os seus agentes e o componente de registro.

3.4.2 - Componentes do sistema de agentes (agência)

A agência proporciona, aos seus agentes, recursos para execução, gerenciamento, transporte e comunicação. Para isoo, executa de seis serviços, além de um sétimo que é opcional:

Segurança *(Security)* - Protege a agência (host) do acesso por agentes não autorizados através do uso de certificados, assim como protege os agentes durante a migração por meio de criptografia SSL. Pode também ativar a criptografia para comunicações entre agências, agentes e outros componentes. Permite, também, ao usuário estabelecer de níveis de segurança de acordo com a identificação da origem do agente.

Registro *(Registration)* - Cada agência registra os agentes presentes visando monitoramento e controle do processamento interno na agência, além de permitir localização mútua entre os agentes, em uma mesma agência, para troca de informações.

Persistência *(Persistence)* - Visa preservar os dados quanto a uma falha do sistema, armazenando periodicamente os estados internos dos agentes. Permite que, após uma falha o sistema retome do último estado salvo, e também, que o agente solicite seja salvo, em um dado momento ou periodicamente.

Gerenciamento *(Management)* - Permite o gerenciamento dos agentes, através de uma interface gráfica de usuário (GUI), podendo os mesmos serem criados, ativados/suspensos, excluídos ou clonados.

Transporte *(Transport)* - Serviço que permite a serialização do estado do agente, a transferência à agência de destino através do serviço de comunicação e a ativação no destino.

Comunicação *(Communication)* - Ativado pelo serviço de transporte, permite a transferência do agente entre agências. Controla também a comunicação entre agentes remotos e entidades não baseadas em agentes, integrando tecnologias cliente/servidor e de agentes móveis. Pode ser síncrona ou assíncrona.

MAFAgentSystem (opcional) - Interface parte do padrão OMG MASIF para aumentar a interoperabilidade entre o Grasshopper e outras plataformas de agentes móveis.

3.4.3 - O componente de registro *(Region Registry)*

Em uma dada região, é permitido que uma agência assuma o papel de registro de toda a região. São mantidos registros de todas as agências e de seus agentes nesta região. Quando da tranferência de agentes, este registro é automaticamente atualizado, permitindo ao administrador do sistema e aos agentes a localização de um agente específico em toda a região.

Possui os seguintes serviços:

Gerenciamento *(Management)* - Serviço responsável pela localização de agentes na região.

Comunicação *(Communication)* - Permite a comunicação entre o componente de registro e agências ou agentes. Equivale ao mesmo serviço em uma agência.

MAFFinder - Interface padrão OMG MASIF para aumentar a interoperabilidade entre o Grasshopper e outras plataformas de agentes móveis.

3.4.4 - Transmissão do agente

Um agente móvel é criado dentro de uma agência. Este termo "agência" define um local no servidor Grasshopper onde os agentes são criados. Pode-se ter mais de uma agência por região.

Usando-se o API da agência pode criar o agente através do método createAgent ( ). O construtor de agente é chamado então para a criação deste novo agente. Este mesmo construtor pode ser chamado novamente após cada procedimento de migração ou cópia do agente. Observa-se que o método init ( ), é chamado uma única vez.

A criação do agente pode se dar de duas maneiras diferentes: através da utilização da interface de usuário (UI) ou pelo método createAgent ( ) constante do API da agência. No primeiro método, o agente recebe seus parâmetros por meio de uma *String array*. O agente então converte os valores nos tipos de dados equivalentes.

Cada agente Grasshopper carrega informações que podem ser acessadas por ele ou por outros. Estas informações são mantidas em uma instância da classe *.ikv.grasshopper.type.AgentInfo*. Quando da criação de um novo agente, por uma agência, uma nova instância de *AgentInfo* é inicializada e transferida ao agente. Para acessar esta instância o agente chama o método *getInfo ( )* que está implementado em sua superclasse *Agent*.

Este objeto *AgentInfo* é parte dos dados do estado do agente. Logo, ao migrar, o agente carrega estes dados consigo. Sua estrutura é criada e inicializada durante a criação do agente.

O primeiro grupo é estabelecido pelo criador do agente, tais como: *codebase* e o *classname*. Outros são automaticamente gerados pela agência na qual o agente é criado, via interface de usuário ou método *createAgent ( )*, tais como o identificador do agente. E um terceiro grupo é gerado dentro do método init ( ), tais como o nome do agente e sua descrição.

A migração de agentes no Grasshopper é classificada como migração fraca, ou seja, além do agente (objeto) são serializados os valores das variáveis. Na migração forte seriam serializados também os valores de pilha e o ponto de onde a execução seria retomada. Esta restrição está associada ao Java, que possui como padrão a migração fraca.

3.4.5 - Protocolo

A comunicação entre os agentes se dá por invocação de métodos Java local ou remota. As comunicações entre as agências e entre as regiões são efetuadas via um ORB (Object Request Broker) proprietário (Grasshopper ORB), socket ou RMI.

> *"A compatibilidade com o MASIF provê uma forte indicação da habilidade de interagir com os ORBs do CORBA"* (Perdikeas, 1999:2006).

Um agente (cliente) de uma agência ao acessar outro agente (servidor), em outra agência, utiliza um proxy que estabelece a conexão entre o cliente e o servidor. (figura 41)

Estando os agentes em agências situadas em uma mesma região, o proxy automaticamente contata o componente de registro para determinar a localização do servidor. Caso o agente servidor se mova, o proxy o localiza através da obtenção da nova localização junto ao componente de registro da região.

A comunicação entre os agentes cliente e servidor pode se dar de modo síncrono ou assíncrono, estático ou dinâmico e *unicast* ou *multicast*.

A utilização de proxies varia nas diferentes versões do Java. Nas versões anteriores ao JDK 1.3, as classes *proxy* têm de ser criadas manualmente por meio do gerador de *stub* do Grasshopper (Stubgen). No JDK 1.3 o agente cliente gerará *proxies* em tempo de execução sem utilizar as classes geradas pelo Stubgen, mesmo que disponíveis.

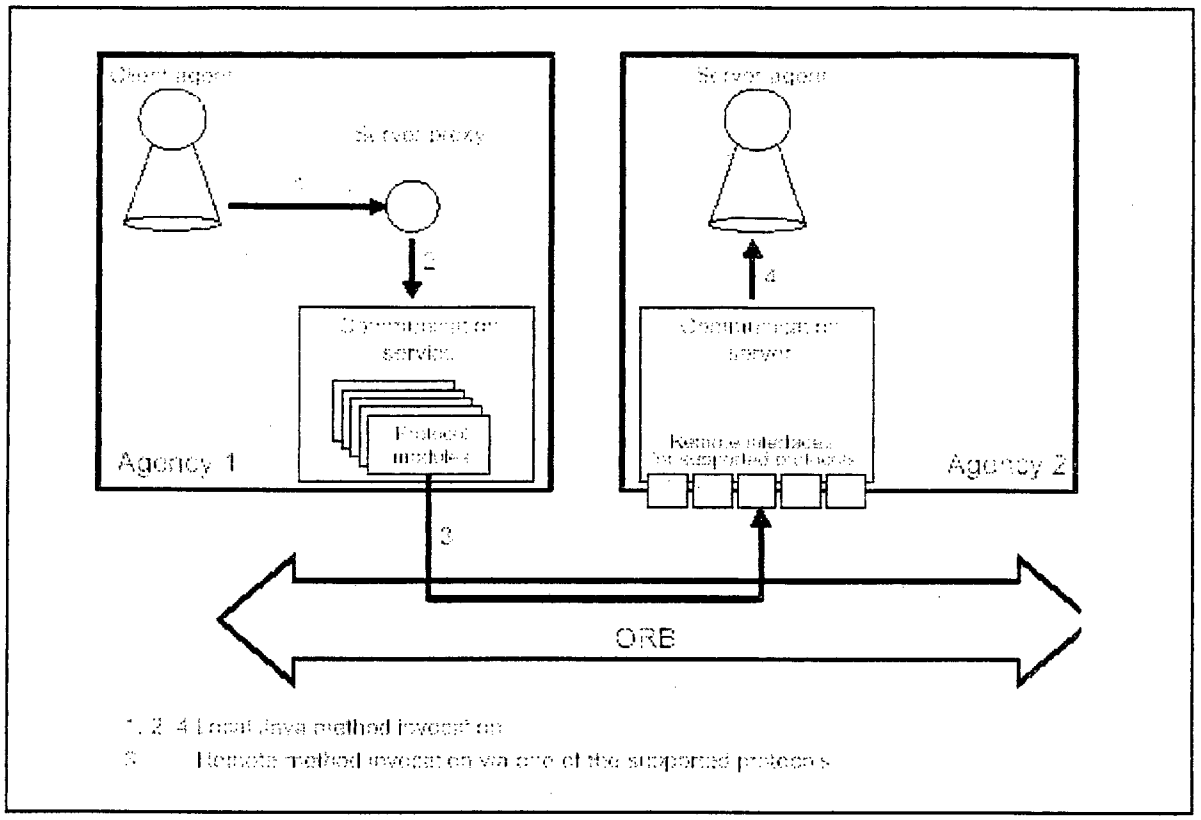


Figura 41 - Comunicação via proxies. (Grasshopper Programmers Guide 2.1)

3.4.6 - Segurança

São considerados os aspectos de segurança ligados à agência (host), aos agentes e à origem do agente.

O agente tem sua origem e autenticidade comprovada por meio de certificados X.509 e assinaturas digitais, protegendo assim o host (agência) contra agentes maliciosos.

A proteção do agente durante sua transmissão é feita por meio de protocolo SSL.

As preferências de segurança são estabelecidas por meio de uma interface gráfica de usuário (GUI).

3.5 - MAP / Università di Catania

A plataforma MAP (Mobile Agent Plataform) foi elaborada com a finalidade de prover as ferramentas básicas para o gerenciamento de agentes móveis em conformidade com as interfaces e funcionalidades do OMG MASIF. (Fig.42)

Neste trabalho foi utilizada o MAP versão 3a compatível com Java 1.2 e Visibroker 4.1.1 e posteriores.

Criada no Departamento de Engenharia de Informática e Telecomunicações da Universidade de Catania/Itália, possui recursos para criação, execução, suspensão, reativação, desativação, reativação e exclusão de agentes. Pode ainda carregar dinamicamente classes presentes em outros hosts.

Por não possuir um ORB próprio, exige a instalação de programa com esta finalidade. Neste trabalho foi adotado o Visibroker 4.5.1 para Java, versão de avaliação, compatível com OMG CORBA, especificação 2.3.

Seus autores consideram que a proposta MASIF não lidou adequadamente com a interação entre agentes e plataformas de agentes em diferentes regiões. Propuseram a inclusão de extensões a estas interfaces de modo a permitir a interoperabilidade entre tais plataformas.

Apesar da compatibilidade com a especificação MASIF, nenhum teste foi efetuado pelos seus autores no sentido de uma troca de agentes com outra plataforma compatível com MASIF.[4]

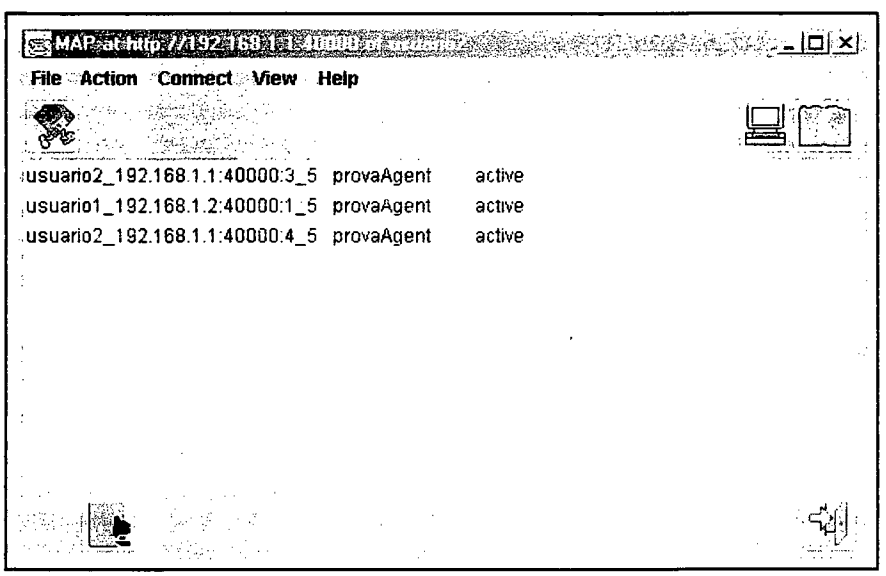


Figura 42 - Interface gráfica MAP.

3.5.1 - Partes principais

Possui cinco partes principais, (Fig.43):

[4] Informação obtida diretamente com o autor do MAP, em trocas de *e-mail* para elaboração deste.

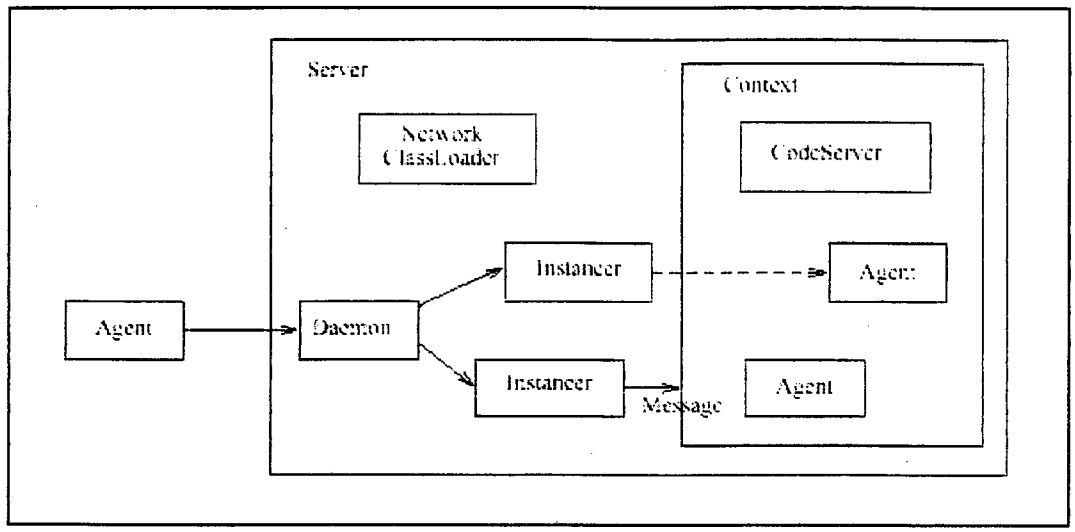


Figura 43 - Arquitetura MAP (La Corte, 1999:7)

Context - Um dos objetos em um servidor MAP, provê um contexto, com conhecimento de todos os agentes presentes e provendo todas as funcionalidades para o gerenciamento dos agentes.

Daemon - Possui um *listener* para uma determinada porta, aguardando para envio de agentes a outro servidor e de mensagens para agentes locais. Cada vez que um *stream* de dados chega, um objeto chamado *Instancer*, instancia o objeto serializado e repassa ao Contexto. O contexto pode então inicializá-lo caso seja um agente ou enviá-lo caso seja uma mensagem.

Network Class Loader (NCL) - Objeto que permite o carregamento dinâmico de classes, mesmo que não presentes no servidor. O NCL busca as classes necessárias em outros servidores, através de questionamentos a seus *Code Servers*. A classe, ao ser localizada, é carregada do servidor remoto e salva nas memórias cache do NCL e do Contexto.

Code Server - Entidade do Contexto que mantém registro das classes disponíveis no servidor local. Através do gerenciamento do NCL, as classes salvas no *Code Server* podem ser transferidas a outros servidores, quando necessário.

Agents - Objetos que executam as atividades designadas através das funcionalidades oferecidas pela plataforma. Um agente pode suspender a si ou outros agentes, reativar um agente desativado, criar um agente, extinguir um agente, migrar a outro servidor.

3.5.2 - Operações fundamentais

O MAP implementa a transferência fraca, ou seja, dos três itens de um agente - o código, o conteúdo das variáveis e pilha - somente o código e as variáveis são enviados.

Através do NCL, uma classe não presente localmente pode ser localizada em outro servidor MAP. Este recurso permite, inclusive, que se crie um agente em outro servidor.

A comunicação entre agentes, embora não especificada no MASIF, a mesma foi implementada no MAP permitindo a troca de mensagens entre agentes de modo síncrono ou assíncrono.

3.5.3 - Transmissão do agente

Criação - Ativada pelo método *runAgent* do Contexto, necessita três parâmetros: o nome da classe, o endereço URL - inclusive a porta, e o nome do servidor MAP onde se encontra a classe. Caso o endereço URL e o nome do servidor Map apontem para o mesmo local, o agente é instanciado e executado no mesmo local. Caso sejam diferentes, o agente é instanciado no servidor citado e transferido para o endereço URL, antes de ser iniciado.

Suspensão - Permite a suspensão da execução de um agente, para posterior reinício do ponto onde parou.

Reinício - Permite a ativação de um agente suspenso anteriormente. Através do método *resume* localiza o agente em uma lista e reativa sua *thread* de execução.

Desativação - Através do método *deactivate*, funciona do mesmo modo que a suspensão, somente serializando o agente e retirando-o da lista para posterior reinício.

Reativação - Através do método *activate*, reativa um agente desativado anteriormente. Funciona de modo similar ao *resume* somente carregando antes os dados.

Exclusão - Através do método *dispose*, pára a execução do agente e cancela seu registro na lista de agentes. Ficando sujeita à coleta de lixo.

Envio - Através do método *go*, o contexto localiza o agente, o serializa, envia ao destino, pára sua *thread* de execução, e exclui a entrada desta *thread* no vetor de agentes. No destino o *Daemon*, através do *Instancer*, chama o método do Contexto para instanciação do agente.

Comunicação entre agentes - Através do método *sendMessage* do Contexto, a mensagem é passada na forma serializada para o *Instancer* do destino, o qual, ao identificar que se trata de uma mensagem, a passa ao agente.

3.5.4 - Segurança

Na versão atual não foram encontrados mecanismos de proteção, seja para proteção da plataforma contra agentes maliciosos ou dos agentes e seus dados.

4 - A INTEROPERABILIDADE DAS PLATAFORMAS

Desde o surgimento dos primeiros computadores com processamento centralizado até a grande expansão das atuais redes de computadores, sempre houve a necessidade de que a informação chegasse ao seu destino com rapidez e confiabilidade, seja para fins de registro ou consulta a banco de dados, seja para possibilitar a realização de outras transações. Com isso muitas tecnologias foram sendo implantadas.

No início dos anos 70, surgiram as primeiras redes locais após o surgimento de micro e minicomputadores. Na década de 80, acompanhando o desenvolvimento das redes de telecomunicações e sua digitalização, ocorreu a interligação das redes, tendo início as primeiras redes públicas. Na década de 90, cresce a interoperabilidade das redes, com a expansão de redes locais pela adoção de modelos cliente/servidor, em substituição ou integrando-se a sistemas *mainframe* e o surgimento de tecnologias de alta velocidade tais como Frame Relay, ATM (Assynchronous Transfer Mode), SMDS (Switched Megabit Data Service), da *Internet*, redes virtuais, etc.

As redes corporativas, à medida que surgiam, eram usadas para variadas atividades que, normalmente, envolviam acesso remoto a informações e comunicação, e, a partir de sua expansão passaram a ser usadas correntemente.

Com a crescente integração de processos entre empresas de naturezas diversas, surgiram necessidades de fluxo de informação do sistema de informática, inclusive de transações online, cada dia mais importantes em numerosos ramos de atividade. Com isso, as questões de interoperabilidade passaram a ter importância crescente, sobretudo face à adoção de arquiteturas e de equipamentos tecnológicamente diferentes, o que dificulta a comunicação destes sistemas, de uma forma correta e eficiente.

Muitas soluções vem sendo propostas, e muitas estão em testes ou em plena utilização. Os resultados tem sido mais satisfatórios ou menos, dependendo do ramo de atividade em que atua a empresa.

A utilização da tecnologia de objetos distribuídos tem sido amplamente utilizada na resolução de problemas que envolvem a necessidade de interoperação entre sistemas.

Uma das propostas, dentro do ramo de objetos distribuídos, é a utilização de agentes móveis, objetos que se movem, de uma rede à outra, para a realização de alguma atividade que lhe foi proposta. O que pode ser a busca de alguma informação ou a efetivação de alguma transação.

Para isso, este objeto possui, além de mobilidade, autonomia e outras habilidades. Muitas vantagens podem ser obtidas com esta tecnologia, tais como, o processamento no local da informação, não ser afetado por falhas na rede e o processamento paralelo através do envio de múltiplos objetos a diferentes locais.

Muitos sistemas de informática vem utilizando plataformas de agentes móveis para busca de informações e realização de transações online, entre diferentes sistemas de uma ou mais empresas.

Muitas plataformas diferentes para agentes móveis, com muitas delas possuindo boa aceitação e penetração no mercado vem sendo usadas. Todavia, uma empresa ou grupo de empresas ao adotar uma destas plataformas não conseguirá trocar informações, com outra empresa, ou grupo de empresas, que adote outra plataforma.

Para que essa troca de informações seja possível, surgiu a necessidade de um padrão que permita a troca de agentes entre estas plataformas, fazendo com que o agente de uma, ao chegar à outra, possa ser recebido e realize as atividades que lhe foram propostas.

Apesar da existência e da adoção de um padrão dessa natureza resolver alguns problemas ligados à interoperabilidade de plataformas, seu desenvolvimento esta envolto em diversas dificuldades. Inicialmente enfrenta reações de ordem comercial. Algo semelhante ao que ocorreu com o sistema elétrico (sobretudo as lâmpadas) no final do século XIX, quando cada empresa tinha suas medidas e especificidades[5].

Outra dificuldade é de ordem tecnológica. Os fabricantes têm que adaptar suas plataformas a exigências externas às quais, parece, muitos não pretendem se sujeitar. Entretanto, tudo leva a crer que não há outro caminho. Na área tecnológica, alguns padrões são estabelecidos por meio de discussões que envolvem centros de pesquisa públicos e privados, universidades, empresas, governos e usuários, e levam à sua posterior adoção. Ou, são estabelecidos por outros mecanismos que podem ser, inclusive, a utilização do poder econômico de determinadas empresas.

---

[5] À época, a opção por componentes de determinada marca levava obrigatoriamente a utilização de lâmpadas do mesmo fabricante, devido a incompatibilidades nos formatos e dimensões das conexões e bulbos.
O estabelecimento de normas e padrões para estes equipamentos possibilitou ganhos aos envolvidos neste setor: consumidores, eletricistas/projetistas, e aos fabricantes. Aos consumidores foi dado o benefício da escolha de quais modelos e fabricantes comprar, levando em conta fatores como preço, qualidade, etc. Os eletricistas passaram a ter a vantagem de montar uma rede elétrica com materiais de quaisquer fornecedores que desejarem, levando em conta o custo-benefício dos itens individualmente, e tendo a certeza de um funcionamento adequado. Os fabricantes tiveram vantagens, pois passaram a ter acesso a qualquer mercado para vender seus produtos, sem a necessidade de conquista de um grande parque instalado. Determinados fabricantes puderam se dedicar ao fornecimento de itens específicos que poderiam vir a ser amplamente utilizados.

Além disso, alguns padrões, apesar de serem devidamente estabelecidos, não são adotados pelos usuários, tornam-se padrões *de jure*, como o padrão OSI/ISO. Enquanto outros adquirem ampla aceitação, tornando-se padrões *de facto*, como o padrão TCP/IP.

Os agentes móveis vêm tendo uma aceitação e utilização crescente, como demonstra a quantidade de plataformas existentes e o constante surgimento de outras. A adoção desta tecnologia poderia ser ainda maior se estas plataformas pudessem interoperar, o que leva à necessidade de elaboração de um padrão, que adotado por todas as plataformas, garantiria a comunicação entre as plataformas e a troca de agentes.

Nesse sentido, dois padrões foram propostos por organizações diferentes. OMG (Object Management Group) e FIPA[6] (Foundation for Intelligent Physical Agents).

A proposta OMG denominada MASIF, propõe a interoperabilidade de algumas funções entre as plataformas, com vistas a permitir o recebimento e a execução de agentes oriundos de outros sistemas, como descrito em outra seção deste trabalho.

4.1 - Etapas dos testes

De acordo com a metodologia definida previamente e tendo disponíveis os equipamentos e programas necessários ao trabalho, definiu-se a seqüência dos testes a serem realizados.

Como equipamentos foram utilizados dois computadores Pentium II sendo um com velocidade de 380 MHz e outro com de 350 MHz, com 94MB e 32MB de memória, respectivamente, com seus periféricos, ligados em rede Ethernet 10Base-T.

Como programas, foram utilizados o Java nas versões 1.1.8, 1.2 e 1.3; e as ferramentas CORBA: Visibroker 4.5.1, OrbixWeb 2000. Além das plataformas.

Em uma **primeira etapa**, foi feita a seleção das plataformas de agentes móveis que seriam utilizadas. Uma vez que a interoperabilidade em questão é entre plataformas que se utilizam de determinada linguagem, e não entre linguagens, optou-se pela utilização de plataformas baseadas em Java, em razão dos motivos descritos anteriormente neste trabalho.

Entre os fatores que levaram à escolha de algumas delas, foram considerados: disponibilidade para instalação; disponibilidade de documentação e/ou manuais de utilização;

---

[6] As especificações FIPA, elaboradas pela Foundation for Intelligent Physical Agents (FIPA) especificam tecnologias para agentes genéricos que visam aumentar a interoperabilidade de aplicações baseadas em agentes com outras aplicações, inclusive, não baseadas em agentes. Procura promover um interoperação funcional e semântica, através da definição de uma linguagem de comunicação, entre agentes denominada ACL (*Agent Communication Language*). As especificações FIPA não são objeto do estudo deste trabalho, maiores informações podem ser obtidas em: http://www.fipa.org

desenvolvimento/utilização não descontinuada; citação em artigos, revistas ou congressos; participação dos autores ou centros de pesquisa nos esforços de padronização MASIF. Aspectos que sinalizaram para a seleção das plataformas Aglets, Voyager, Concordia, Grasshopper e MAP, sendo que esta última foi incluída pela alegada compatibilidade com o MASIF.

A **segunda etapa** consistiu na instalação e testes de funcionamento das plataformas incluindo a análise dos programas, sua documentação e revisão da literatura visando determinar sua compatibilidade com o MASIF.

Uma das principais necessidades foi o estabelecimento de quais versões do Java utilizar, pois excetuando-se o Voyager, todas as outras plataformas requerem versões específicas. Apesar da possibilidade de utilização de versões diferentes do Java em uma mesma máquina, por questões de simplificação diferentes versões foram instaladas em diferentes máquinas.

A utilização de algumas plataformas em quaisquer versões do JVM poderia ser feita através da criação de classes apropriadas. Muitas tentativas neste sentido, em especial com o Aglets, foram feitas por vários pesquisadores ao redor do mundo conforme duas das mais abrangentes listas de discussão do Aglets na Internet (aglets-users@lists.sourceforge.net; aglets-developer@lists.sourceforge.net). Apesar desta solução resolver problemas localizados, não tem funcionado para utilizações mais abrangentes.

Logo optou-se pela utilização de versões diferenes do JVM nas máquinas.

Na máquina 1, foram instalados o JDK 1.1 8 e as plataformas Aglets 1.1.0 e Voyager 4.0. E na máquina 2, foram instalados o JDK 1.2 e as plataformas Voyager 4.0, Concordia 1.1.7 e Grasshopper 2.2.

Após a instalação e testes do JDK nas máquinas, passou-se à instalação e aos testes das plataformas nas máquinas.

Os testes consistiam da inicialização de duas plataformas iguais em uma mesma máquina e em portas distintas. Uma delas deveria inicializar um agente e enviá-lo à outra máquina, que deveria recebê-lo e enviar de volta, de maneira automática ou com intervenção manual.

Para realizar o teste, em cada plataforma, foram escolhidos agentes entre os exemplos incluídos no pacote, uma vez que não faz parte do escopo deste trabalho a construção de um agente.

De um modo geral, foram feitos testes com três tipos de agentes, em cada plataforma.

No primeiro teste, um agente "Hello World" deveria ir até a máquina de destino e colocar uma mensagem na tela.

No segundo teste, um agente "Bumerangue" deveria ir até a máquina de destino e retornar automaticamente ou solicitar confirmação para retorno.

O terceiro teste teve duas variações conforme a plataforma. Na primeira, um agente deveria obter informações sobre horário e tarifas de vôo entre duas cidades, em uma determinada data. Na segunda, um agente deveria efetuar uma consulta do saldo bancário de um determinado cliente, incluir um depósito e obter o novo saldo.

Após a realização destes testes que comprovaram o funcionamento correto de cada uma das plataformas, foram completadas as análises dos programas, sua documentação, para determinar as plataformas, em conformidade com o MASIF.

Durante esta etapa, foi verificada, pela análise das classes e confirmada pela literatura, a incompatibilidade das plataformas, exceto Grasshopper, com a especificação MASIF.

A utilização de ORBs proprietários, não compatíveis com o OMG CORBA, dificulta a implementação de operações para comunicação entre as plataformas.

O Aglets utiliza o protocolo ATP para a transmissão de agentes e mensagens. O Aglets utiliza uma interface interna para comunicação entre a *runtime* e o sistema de comunicação, mas não disponibiliza esta interface externamente com uma interface CORBA. Conforme documentação do Aglets, uma camada de transporte compatível com CORBA/IIOP deverá tornar-se disponível em versões futuras.

O Voyager possui um ORB com habilidades para manusear objeto Java, todavia não é uma ORB compatível com o ORB padrão da OMG (CORBA Specification 2.3).

O Concordia utiliza TCP/IP e não impõe um protocolo próprio. Utiliza o RMI do Java e uma classe especial denominada *ClassLoader*. Esta classe opera de modo similar ao de um navegador carregando uma applet Java.

O Grasshopper utiliza um ORB próprio baseado no RMI do Java. Suporta ainda os protocolos Socket, CORBA/IIOP e RMI. Conforme a literatura disponível, é a única plataforma comercial compatível com o MASIF.

Após os teste de funcionamento das quatro plataformas originalmente escolhidas, foi identificada uma outra plataforma denominada MAP desenvolvida pela Università di Catania/Itália compatível com a proposta MASIF.

Esta plataforma foi então instalada na máquina 2, por exigir Java 2, e realizados, com sucesso, os testes de funcionamento executados anteriormente com as outras plataformas.

Entre as plataformas testadas, somente a Grasshopper e MAP são compatíveis com a proposta MASIF (quadro 4). Passou-se então a etapa de testes de interoperabilidade.

Nesta **terceira etapa**, o Java 2 foi instalado também na máquina 1, passando as duas máquinas a possuir a mesma versão instalada. Procedeu-se à instalação das duas plataformas selecionadas nas duas máquinas.

Em função dos testes que visavam determinar a interoperabilidade utilizarem ORB's compatíveis com o CORBA especificação 2.3, a versão Java das máquinas foi novamente alterada para Java 2 SE v.1.3 por possuir esta, as classes OMG CORBA (por ex: org.omg.corba) em sua instalação *default*.

Foi também instalado o Visibroker 4.5.1 que efetivamente atuaria como ORB para plataforma MAP e Grasshopper.

Variados testes foram feitos visando determinar se há interoperabilidade, ou não, destas plataformas.

4.2 - Testes Grasshopper

A ativação do Grasshopper exigiu as seguintes etapas:

A- Ativação (opcional) do Region Registry. O Region Registry atua como um registro de todos os servidores (agências) e agentes presentes na região em que se encontra.

O acionamento da plataforma Grasshopper, para registro de uma região, se dá através do acionamento do programa de lote grasshopper.bat[7] como segue:

Grasshopper r -n MinhaRegiao -gui -tui

- r - indica acionamento do componente de registro de região.
- -n - indica que o próximo parâmetro é o nome a ser dado a esta região.
- MinhaRegiao - nome atribuído à região.
- -gui - aciona a interface gráfica de usuário.

---

[7] O programa de lote grasshopper.bat contém:

..

java -cp %..\lib\gh.jar;..\lib\jndi.jar;..\lib\ldap.jar;%CLASSPATH%" de.ikv.grasshopper.Grasshopper %1 %2 %3 %4 %5 %6 %7 %8 %9

..

	Aglets	Voyager	Concordia	Grasshopper	MAP
Ciclo de vida (agente)					
Clonagem	Sim	Sim	Sim	Sim	Não
Criação remota	Sim	Sim	Não	Sim	Sim
Copia	Não	Não	Não	Sim	Sim
Persistência	Explícita; Limitada.	Explicita para um período arbitrário.	Implícita	Implícita; Explícita para períodos arbitrários.	Não
Duração da vida	O agente vive para sempre; ou até que seja chamado o método de remoção do agente.	Até que seja chamado o método de remoção do agente ; ou até a ocorrência de um evento determinado.	Até a chamada do método completedItinerary ( ), pela plataforma.	O agente vive para sempre; ou até que seja chamado o método de remoção do agente.	Até que seja chamado o método de remoção do agente.
Segurança					
Autenticação	Autenticação de domínios e algoritmos simétricos.	Não.	Identidade do usuário criptografada com algoritmos simétricos.	Através da utilização de certificados X.509	Não
Controle de acesso	Configurável através de interface gráfica de usuário.	Através do uso ou extensão do método Voyager SecurityManager	Permissões armazenadas em um arquivo de preferências.	Configurável através de interface gráfica de usuário ou extensão do método do gerenciador de segurança	Não
Integridade transmissão	Algoritmos simétricos para detecção de alterações indevidas nos agentes.	Não.	Protocolo SSL.	Protocolo SSL.	Não
Arquitetura					
Protocolos de comunicação	TCP/IP e RMI.	ORB proprietária e suporte à migração de objetos simples Java.	RMI.	TCP/IP, RMI, CORBA IIOP, ORB proprietário.	CORBA IIOP
Compatibilidade MASIF	Não.	Não	Não.	Sim.	SIm.
Modelo de programação do agente	Estendendo-se a classe base aglet	Estendendo-se a classe base voyager	Estendendo-se a classe base concordia	Estendendo-se a classe base grasshopper	Estendendo-se a classe base map

- -tui - aciona a interface de texto do usuário.

B- Ativação da(s) agência(s), que opcionalmente podem ser registradas no Region Registry. As agências de uma determinada região, registradas em um Region Registry, podem estar em hosts (máquinas) diferentes.

O acionamento de uma agência é realizado através do acionamento do mesmo programa de lote, com outros parâmetros:

Grasshopper a -n MinhaAgencia -r socket://192.168.1.1:7020/MinhaRegiao -gui -tui

- a - denota criação de agência.
- -n - indica que o próximo parâmetro é o nome dado à agência.
- MinhaAgencia - nome a ser atribuído à agência.
- -r - indica que a agência deve ser registrada junto à região indicada no próximo parâmetro.
- Socket://192.168.1.1:7020/MinhaRegiao - indica protocolo, endereço IP, número de porta e nome a região onde a agência deve ser registrada.
- -gui - aciona a interface gráfica de usuário.

Em uma das máquinas foi inicializado um Region Registry e duas agências. Agentes foram criados em uma agência e enviados à outra e a seguir enviados de volta.

Na outra máquina foi inicializado outro Region Registry e criadas também duas agências registradas junto a este Region Registry. Agentes foram criados nestas duas agências e trocados entre as mesmas.

Todavia não houve modo de agentes criados em uma determinada região migrarem para agências em outra região, ou seja, os agentes estão restritos à região em que foram criados. Para comprovação deste fato outro teste foi realizado.

Foi criada uma terceira agência em uma das máquinas que, desta vez, foi registrada no *Region Registry* presente na outra máquina. Deste modo, os agentes criados nesta agência conseguiam migrar para as agências, do mesmo *Region Registry*, presentes à outra máquina a vice-versa. Esta terceira agência continuou, como era de se esperar, incapaz de comunicar-se com agências na mesma máquina registradas em outra região.

A plataforma Grasshopper foi incapaz de enviar um agente para fora da região onde foi criado, o que não está em desacordo com a especificação MASIF.

Analisando-se a especificação MASIF, pode-se perceber que há o controle dos servidores (denominados agências, no Grasshopper) e agentes presentes em uma determinada região, não se tendo nenhuma informação sobre outras regiões ou agentes presentes nestas outras regiões.

Estes testes entre regiões no Grasshopper foram feitos com e sem a utilização de ORB's. A utilização de alguns dos agentes-exemplo incluídos no pacote não exigiam a utilização de ORB's. Em outros foi necessária a utilização de ORB's. Para a utilização de ORB's foi acionado o *daemon* do ORB (osagent, no visibroker), e incluídos os parâmetros do ORB (tais como endereço IP e número de porta) no comando de ativação da plataforma[8], incluído no programa de lote, conforme documentação fornecida junto com a plataforma e com orientações do suporte Grasshopper, conforme informações constantes no site do produto. E em conformidade com a documentação do Visibroker.

Posteriormente foram repetidos todos os testes utilizando-se o ORB Orbix 2000, uma vez que este foi citado na documentação do Grasshopper e contém em exemplo no site Grasshopper. Os mesmos resultados foram encontrados.

4.3 - Testes MAP

A ativação do MAP exigiu as seguintes etapas:

A- Ativação do daemon osagent (Visibroker).

B- Registro da região através do acionamento do programa de lote regd.bat[9], como segue:

regd NomedaRegião

- NomedaRegião - Nome da região a ser criada. Previamente estabelecido no arquivo attrib.txt juntamente com endereço IP e número de porta.

C- Ativação da plataforma MAP através do acionamento do programa de lote mapd.bat[10], como segue:

---

[8] O programa de lote grasshopper.bat com os parâmetros do ORB:

..

java -Xbootclasspath/p:%CORBA_CLASSES% -D<objeto Naming Service do ORB=endereço IP> <objeto listener do ORB=nº porta> de.ikv.grasshopper.Grasshopper %1 %2 %3 %4 %5 %6 %7 %8 %9

..

[9] O programa de lote regd.bat:

..

vbj map.finder.Region %1 %2

..

[10] O programa de lote mapd.bat:

..

Set MAP=%Caminho das classes MAP%

mapd NomedaRegião

- NomedaRegião - Nome da região onde ao servidor iniciado vai ser registrado.

Em uma das máquinas foi inicializado o daemon osagent, registrado uma região e inicializado um servidor nesta região. A seguir, na outra máquina foi ativado o daemon osagent e inicializado um outro servidor que foi registrado junto à região presente na primeira máquina, ao invés de criar-se outra região. Como era de se esperar, o agente criado em um servidor foi enviado com sucesso ao outro servidor localizado em outra máquina, porém registrado na mesma região.

A plataforma MAP inclui uma interface extra, denominada ExtendedMAFFinder (Fig.44), criada pelos autores e não previsto no MASIF que a torna capaz de enviar agentes atráves de regiões diferentes o que foi verificado pelos testes a seguir.

```
interface ExtendedMAFFinder:MAFFinder {
  void add_location_agent(
    in Name agent_name,
    in Location region_name) raises(NameInvalid);

  void update_location_agent(
    in Name agent_name
    in Location region_name) raises(EntryNotFound);

  void remove_location_agent(
    in Name agent_name) raises(EntryNotFound);

  Locations register_region_agent(
    in Name agent_name,
    in Locations region_names,
    in Location agent_location,
    in AgentProfile agent_profile) raises(NameInvalid);

  void unregister_region_agent(
    in Name agent_name,
    in Locations region_names,
    in boolean isend) raises(EntryNotFound);

  ExtendedMAFFinder get_ExtendedMAFFinder(
    in Location region_name) raises(FinderNotFound);
};
```

Figura 44 - Interface ExtendedMAFFinder

Em uma das máquinas foram inicializados o *daemon* osagent, em seguida o registro da região e a seguir a plataforma, conforme etapas descritas anteriormente. A seguir, o mesmo foi efetuado em outra máquina, ficando-se então com duas regiões distintas. Em cada uma destas regiões, foi inicializado um servidor de agentes.

---

vbj map.core.Main %MAP% %1 %2 %3 %4

Um agente foi então criado em um servidor pertencente a uma região, e enviado ao servidor registrado na outra região. O agente foi enviado pela origem, recebido e inicializado pelo destino corretamente.

O servidor de destino foi capaz ainda de pausar o agente, reativá-lo e enviá-lo de volta ao servidor de origem, situado em outra região.

Entretanto, um agente ao ser enviado à outra região teve seu controle de localização perdido. Não se sabe, do ponto de vista da região de origem, se o agente continuou sua execução, foi suspenso ou mesmo extinto do servidor de destino.

Outro problema ocorre, como previsto na literatura, do seguinte modo. Um agente (1) ao requisitar um serviço a outro agente (2) em uma determinada região, pode migrar antes de receber os dados solicitados a este outro agente (2). O agente (2), quando retornar com os dados, não localizará o agente (1), uma vez que o controle de agentes da região não possui mais a referência ao agente (1). O que impede a região de reencaminhar o agente (2) ao agente (1). Mesmo que se considere o envio do agente (2) à mesma região para onde se destinou o agente (1) não se saberá se o mesmo o recebeu ou mesmo se ainda estará naquela região.

Foram ainda feitos testes com vistas a registrar-se um servidor (agência) do Grasshopper junto à uma região do MAP e vice-versa, para a troca de agentes entre servidores registrados em uma mesma região ou mesmo em regiões diferentes. O que foi impossibilitado de se testar uma vez que o código fonte da plataforma MAP não estava disponível. E mesmo que estas classes estivessem disponíveis, uma possível solução seria a implementação de classes específicas para a troca de agentes entre o Grasshopper e o MAP.

4.4 - Resultados dos testes

Os testes feitos para determinação de quais plataformas para agente móveis eram compatíveis com o MASIF comprovaram que das cinco plataformas selecionadas, somente duas, Grasshopper e MAP, mostraram-se compatíveis.

Mesmo sendo compatíveis com a especificação MASIF, não houve como uma trocar agentes móveis com a outra. Tal fato ocorreu pela impossibilidade de envio ou recebimento de agentes de outras regiões no Grasshopper.

A plataforma MAP foi capaz de enviar e receber agentes para outras regiões, todavia isto somente foi possível através da criação de uma interface extra (não prevista quando da criação ou revisão da proposta MASIF.

Esses resultados são referendados pela literatura que mostra que a interoperabilidade das diferentes plataformas de agentes móveis não foi, ainda conseguida.

Enquanto não surgem saídas eficazes e simples para se obter essa interoperabilidade, algumas soluções estão sendo implementadas para a resolução de problemas localizados, caso a caso.

Uma destas soluções, descrita como solução para interoperabilidade de diferentes plataformas consiste basicamente em uma interface a ser instalada nas máquinas de origem e destino que queiram trocar agentes entre si e que passam a trocar um tipo de agente definido nesta interface (Fig.45).

Essa interface foi desenvolvida para atender a necessidades imediatas, localizadas, e, em muitos casos, realizar a interoperabilidade entre determinadas e diferentes plataformas de agentes móveis. Por exemplo, uma plataforma Aglets desejando enviar um agente móvel a uma plataforma Voyager, utilizaria a interface criada com esta finalidade. Esta interface interceptaria o envio ao detectar que se trata de um envio à plataforma Voyager, criaria um agente de tipo próprio que seria então enviado à interface par no destino. Esta por sua vez ao receber o agente, o "transformaria" em um agente compatível com Voyager, possibilitando a interoperação.

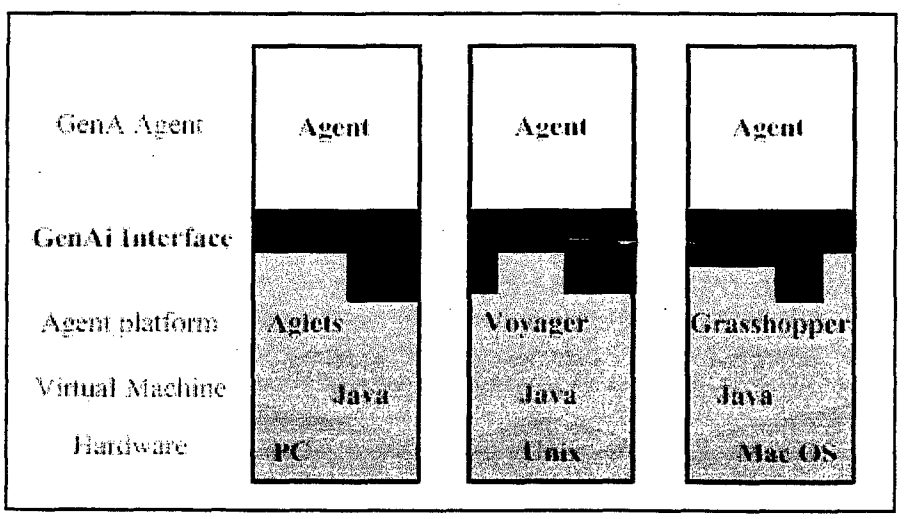


Figura 45 - Interface GenAi. (Magnin, 1999:3)

Como bom exemplo desta solução pode-se citar a plataforma GenA, criada no Centre de Researche Informatique de Montreal/Canadá, desenvolvida visando três plataformas baseadas em Java: o Aglets, o Voyager e o Grasshopper. Nestas a interface instalada

denominada GenAi cria agentes denominados genéricos para troca de agentes entre as interfaces deste tipo e estas plataformas.

Apesar da alegada interoperabilidade, estas interfaces ao serem instaladas se tornam uma camada extra que intercepta a comunicação da plataforma na qual esta instalada e envia o agente para a camada par desta interface instalada na máquina de destino. O que apesar de proporcionar alguma interoperabilidade, não a proporciona nos moldes do MASIF.

Outro fato é que o código destas interfaces tem de estar previamente instaladas tanto na plataforma de origem como na de destino. E para tornar-se compatível com outras plataformas, faz-se necessária a elaboração de classes com esta finalidade e inclusão delas no pacote GenAi. O que gera outros problemas tais como, necessidade de atualização de versão sempre que outra plataforma tornar-se compatível.

Esta solução, apesar de seus méritos na resolução de muitos problemas de interoperabilidade, gera outros problemas. Na verdade, soluções que levem em conta um número limitado de plataformas, levarão alguns *host*, com o passar do tempo, a ter muitas interfaces para interceptação de operações com agentes diferentes. O quê, por questões de recursos computacionais e/ou de segurança pode não ser desejável. Haveria, ainda, a possibilidade de conflitos, uma vez que numerosas e diferentes interfaces viriam a coexistir em um mesmo *host* e a compatibilidade, entre estas, seria de difícil implementação. Além disso, não deveria ser imposto, a um host, a instalação de classes que não sejam as dos agentes que chegam.

A intenção demonstrada pelo autor da interface GenAi de que sua implementação poderia resolver a questão da interoperabilidade não é muito adequada, pois tenta impor uma solução proprietária, a qual encontra muitas resistências por parte dos interessados.

Uma solução deste tipo para que venha a ser adotada deveria partir, ou ser adotada, por um grupo que represente um grande conjunto de interessados na questão da interoperabilidade.

Enfim, além de dificuldades comerciais, para se implantar a interoperabilidade, as de ordem técnica exigirão esforços de desenvolvimento para sua superação. Serão no entanto, indiscutíveis os benefícios que uma interoperabilidade poderá trazer aos usuários de plataformas de agentes móveis, sobretudo porque se sabe que, a cada dia, aumenta o número de empresas e de pessoas que buscam se valer de dados que estão disponíveis atráves plataformas alheias ao seu domínio

CONCLUSÕES

A utilização de agentes móveis mostra-se muito promissora como instrumento de captação de dados e traz benefícios aos usuários. Estes, quando contam com a interoperabilidade de diferentes plataformas, maximizam os ganhos operacionais das redes em: eficiência; economia de recursos pela execução em um nó da rede de cada vez; redução de tráfego, pela interação autônoma assíncrona; robustez e tolerância a falhas; suporte a ambientes heterogêneos; extensibilidade de serviços online; e facilidade de atualização de programas.

Apesar desses benefícios, a interoperabilidade das plataformas encontra-se em fase inicial de desenvolvimento. Há problemas de diversas ordens impedindo o estabelecimento do que será um grande avanço no intercâmbio de agentes móveis. A padronização OMG MASIF ainda não tem, como adeptos efetivos, grandes fabricantes de plataformas que, com operações de impacto no mercado atual, não trocam seus agentes.

Nesta dissertação buscou-se analisar a especificação para padronização de agentes móveis OMG MASIF, inclusive, com a utilização de plataformas compatíveis com esta especificação.

Foi realizada neste trabalho, em um primeiro momento, a seleção das plataformas que, segundo indícios, seriam compatíveis com o MASIF. E em um segundo momento, as plataformas que se mostraram compatíveis com o MASIF foram submetidas a testes para se verificar os níveis de interoperabilidade proporcionados pela especificação MASIF.

Nos resultados dos testes de verificação da compatibilidade, comprovou-se que, entre as diferentes plataformas analisadas, apenas uma de uso comercial - Grasshopper - e outra acadêmica - MAP - mostraram-se compatíveis com a especificação MASIF. Mesmo entre essas duas não foi possível a troca de agentes, uma vez que a plataforma Grasshopper não permitiu o envio de agentes para fora da região na qual foram criados.

O que expôs uma deficiência do MASIF, que na visão do autor, é uma das causas primordiais que previnem uma expansão mais rápida da adoção do MASIF e conseqüentemente da interoperabilidade desejável.

O MAP, mesmo sendo compatível com o MASIF, somente conseguiu enviar agentes para outras regiões devido a uma extensão - não especificada no MASIF - criada com essa finalidade. Mesmo assim, o agente ao ser enviado para outra região, ficou fora do controle de sua região de origem que perde a sua referência - sem saber se ele continua em exercício, se foi suspenso ou extinto.

Essa ausência de interoperabilidade das diferentes plataformas apresenta-se como um empecilho ao acesso a uma vasta quantidade de informações, já disponíveis para algumas plataformas. O que, certamente, tem impedido uma ainda maior integração de empresas e processos.

Para necessidades imediatas de interoperação, uma solução proposta na literatura e já utilizada é a elaboração de interfaces para uso em plataformas específicas que, apesar de resolver alguns problemas mais imediatos, não se constitui em uma solução definitiva.

A compatibilidade de uma plataforma com a especificação MASIF independe de sua implementação, exigindo-se apenas a presença de duas interfaces: MAFAgentSystem que define operações de gerenciamento das atividades do agente; e MAFFinder que define operações para nomeação e localização de agentes e sistemas de agentes dentro de uma região.

A literatura sinaliza que a adoção plena do MASIF exigirá padronização de uma interface ou mesmo a adição de funcionalidades à interface MAFFinder que permita um controle de todos os agentes destinados a outras regiões ou oriundos de outras regiões, nos moldes do controle de localização do MAFFinder para transações intra-regiões. Isso permitiria a movimentação e rastreamento completo dos agentes entre diferentes regiões.

No contexto atual, a utilização dos agentes móveis já se mostra promissora, no entanto, sua generalizada aceitação e ampla utilização irão depender fortemente de estabelecimento de padrões que, realmente, possibilitem a interoperabilidade das atuais e futuras plataformas de agentes móveis. Uma realização que foi tentada pelo OMG, ainda sem sucesso, conforme estudos e testes efetuados com as plataformas MAP e Grasshopper, tidas como compatíveis com as especificações MASIF.

GLOSSÁRIO

ACL (*Agent Communication Language*): linguagem de comunicação entre agentes, desenvolvida pela DARPA. Consiste do formato KIF e da linguagem KQML.

ANSI (*American National Standards Institute*): organização americana, ligada a ISO, responsável pela padronização de redes de computadores e outros assuntos. São ligadas a ela dentre outros a TIA (*Teleccomunications Industrie Association*); ISA (*Instrument Society of America*); SME (*Society of Manufacturers Engineers*); IEEE (*Institute of Electrical and Electronics Engineers*).

API (*Application Program Interface*): Interface de Programa Aplicativo é programa de interface entre aplicativo em utilização pelo usuário e o sistema operacional ou ambiente. Executa tarefas de compilação e link-edição transformando o código recebido em código compacto e apropriado ao sistema operacional ou ambiente.

ASCII (*American Standard Code for Information Interchange*): Código Padrão Americano para o Intercâmbio de Informações, inicialmente um código de sete bits (e o oitavo bit era conhecido como bit de paridade) usado para representar 128 símbolos, incluindo todos os caracteres alfanuméricos básicos, com bits de dados; formalizado pelo Instituto Americano de Padrões, porém mais tarde aprimorado pela IBM; agora um sistema de oito bits que descreve 256 símbolos.

ATM (*Assynchronous Transfer Mode*): Modo de Transferência Assíncrono é uma tecnologia baseada na transmissão de pequenas unidades de informação de tamanho fixo e formato padronizado, denominadas células. Células são transmitidas através de conexões com circuitos virtuais. Suporta diferentes serviços para satisfazer aos requisitos exigidos pelos diferentes tipos de tráfego, a altas velocidades de transmissão.

***Broadcasting*:** conexão na qual os dados são enviados a um servidor de *broadcast* para que seja difundido para outros pontos da rede de uma maneira paralela (simultânea).

Cache: tipo de memória de alta velocidade que contém os dados e instruções mais recentes acessados pela CPU.

Camadas: organização da maioria das redes visando reduzir sua complexidade, são organizadas para que uma ofereça determinados serviços a outra, ocultando os detalhes de sua implementação.

CCITT (*Comité Consultatif International Télégraphique et Téléphonique*): antiga denominação do ITU-T de 1956 a 1993. Em 1° de março de 1993, foi reorganizado para se tornar menos burocrático e teve seu nome alterado para refletir as novas funções.

CORBA (*Common Object Request Broker Arquitecture*): arquitetura e infra-estrutura aberta na qual aplicações de computador usam para se comunicar em redes. Usando o protocolo padrão IIOP (*Internet* Inter-ORB *Protocol*), um programa baseado um CORBA de qualquer

fabricante em praticamente qualquer computador, sistema operacional, linguagem de programação e rede podem interoperar com um outro programa baseado em CORBA.

CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*): protocolo utilizado na sub-camada MAC de redes locais (LAN's), na qual todas as máquinas da rede possuem acesso ao meio. Quando duas ou mais máquinas o fazem simultaneamente, ocorre uma colisão. Cada uma então aguarda um tempo aleatório e tenta retransmitir. O padrão IEEE 802.3 (Ethernet) é uma de suas versões.

Datagramas: pacotes independentes da organização enviados pela rede, sem conexão.

DNS (*Domain Name Service*): esquema de gerenciamento de nomes, hierárquico e distribuído. Define a sintaxe dos nomes usados na *Internet*, regras para delegação de autoridade na definição de nomes, um banco de dados distribuído que associa nomes a atributos (entre eles o endereço IP) e um algoritmo distribuído para mapear nomes em endereços. Especificado nas RFC's 882, 883 e 973.

Ethernet: padrão ANSI/IEEE 802.3 (ISO 8802.3) para redes em barra que utilizam protocolo de acesso CSMA/CD.

Host: máquina em uma rede cuja finalidade é executar os programas do usuário.

HTML (*HyperText Markup Language*): linguagem especializada em hipertextos, voltada para aplicações *Web*.

Http (*HyperText Transfer Protocol*): protocolo de transferência padrão da *Web*. Cada interação consiste de um conjunto de solicitações dos *browsers* (navegadores) aos servidores e um conjunto de respostas que retornam no caminho inverso. Semelhante ao FTP, utilizando-se de uma conexão TCP para cada transferência.

IEEE (*Institute of Electrical and Electronics Engineers*): organização profissional que publica uma série de jornais e promove diversas conferências a cada ano. Possui um grupo de padronização que desenvolve padrões nas áreas de Engenharia Elétrica e Informática. O padrão 802.3 do IEEE para as redes locais é o padrão mais importante adotado pelas LAN's.

***Internet*:** enorme rede que liga muitos computadores científicos, educacionais, de pesquisa, educacionais do mundo, e também redes comerciais; também chamada de *Net* (rede).

IP (*Internet Protocol*): protocolo da camada de rede. Sua tarefa é fornecer a melhor forma de transportar datagramas da origem ao destino determinado por um endereço IP, independente de estas máquinas estarem na mesma rede ou em outras redes intermediárias.

ISO (*International Organization for Standardization*): organização internacional voluntária fundada em 1946, que produz padrões internacionais. Seus membros são as organizações nacionais de vários países, entre as quais podem ser citadas: DIN (Deutsches Institut for Normung)/Alemanha; BSI (British Standard Institute)/Grã-Bretanha; ANSI (American National Standards Institute; ABNT (Associação Brasileira de Normas Técnicas). Publica padrões sobre uma vasta gama de assuntos. Para a área de telecomunicações, trabalha em conjunto com o ITU-T, da qual é membro.

ITU (*International Telecommunication Union*): órgão membro das Nações Unidas responsável pela padronização das telecomunicações. Possui três divisões: ITU-R para radiocomunicação; ITU-T para telecomunicação; e ITU-D para desenvolvimento.

ITU-T (*International Telecommunications Union - Telecommunications*): órgão do ITU responsável pela padronização das telecomunicações internacionais, anteriormente denominado de CCITT. Possui cinco classes membro: administrações denominadas PTT's (órgãos estatais gestores de telecomunicações); operadores privados reconhecidos; organizações de telecomunicações regionais; organizações científicas e fornecedores de telecomunicações; e outras empresas interessadas.

Java: linguagem de programação desenvolvida pela Sun Microsystems, originalmente concebida para ser usada com a televisão interativa nos Estados Unidos. Orientada a objetos, suporta multiprocessamento, possui ainda coleta de lixo automática da memória e uma série de recursos para a produção de interfaces gráficas, aplicações para redes e processamento distribuído.

KIF (*Knowledge Interchange Format*): formato de troca, que conjuntamente com o KQML faz parte da linguagem ACL.

KQML (*Knowledge Query and Manipulation Language*): linguagem de comunicação entre agentes que conjuntamente com o KIF faz parte da ACL, desenvolvida pela DARPA.

LAN (*Local Area Network*): Rede Local - sistema de computadores localizados relativamente perto uns dos outros e conectados por meio de cabo ou outro meio, de modo a permitir que cada usuário possa cooperativamente processar informações e compartilhar recursos.

MAN (*Metropolitan Area Network*): Rede Metropolitana - duas ou mais redes locais conectadas, geralmente cruzando uma ampla geográfica que abrange uma cidade ou região metropolitana.

***Middleware*:** *software* que conecta duas aplicações. Pode-se ter vários *middleware* para ligar as duas aplicações. Exemplos: sistemas de acesso a banco de dados; ORB (*Object Request Broker*).

Objeto: unidade independente definida dentro de uma instrução de programação orientada a objetos; contém tanto dados quanto funções.

ORB (*Object Request Broker*): componente em CORBA que age como um *middleware* entre clientes e servidores, um cliente pode requisitar um serviço não sabendo nada sobre quais servidores estão ligados à rede. Os vários ORB recebem solicitações, as repassam para os servidores apropriados e retornam com o resultado para o cliente.

OSI/ISO (*Open Systems Interconnection*): *International Organization for Standardization*, modelo de referência *de jure*, também denominado RM-OSI (*Reference Model - Open System Interconnection*), objetivo do padrão internacional 7498-1:1984, composto de sete camadas, que fornece uma base comum que permita o desenvolvimento coordenado de padrões para a interconexão de sistemas abertos.

PDA (*Personal Digital Assistants*): Assistente Pessoal Digital - computador portátil, aproximadamente do tamanho de um calculador pessoal.

RFC (*Request for Comments*): documento necessário para que um protocolo se torne um padrão *Internet*. As RFC's podem ser acessadas por qualquer pessoa conectada à *Internet*. Após decorrido determinado período o protocolo é declarado pelo IAB (*Internet Activity Board)* como um *Internet Standard* (padrão *Internet*).

RMI (*Remote Invocation Method*): método da linguagem Java para busca de referência a objetos remotos.

Tcl (Agent-Tcl; Safe-Tcl): linguagem *script* utilizada no programa Telescript da General Magic/EUA.

TCP (*Transmission Control Protocol*): protocolo de controle de transmissão utilizado na camada de transporte. Orientado à conexão, confiável, com entrega de um fluxo de dados sem erros.

TCP/IP (*Transmission Control Protocol/Internet Protocol*): modelo de referência *de facto* Para interconexão de redes composto de cinco camadas com o objetivo de conectar várias redes ao mesmo tempo através da implementação do protocolo de nível de rede IP e do protocolo de nível de transporte TCP.

***Thread*:** Uma parte de um programa que pode executar independentemente de outras partes. Sistemas que permitam *multi-threading* permitem aos programadores criar programas em que algumas dessas partes possam executar concorrentemente.

UDP (*User Datagram Protocol*): protocolo não confiável, sem conexão, utilizado para o envio de mensagens que não necessitem de controle de erro ou de fluxo. Normalmente utilizado onde rapidez é mais necessário do que exatidão nos dados enviados.

URL (*Uniform Resource Locator*): atribuição de uma página na *Internet*. Composta de três partes. O protocolo (por exemplo: http), o DNS (nome atribuído à página) e o nome do arquivo a que se refere (por exemplo: index.html).

WAN (*Wide Area Network*): Rede de Longa Distância ou remota - duas ou mais redes locais conectadas, geralmente cruzando uma ampla área geográfica.

BIBLIOGRAFIA

ALVES, M. B. M. & ARRUDA, S. M. (2000) Como Fazer Referências Bibliográficas. Florianópolis: UFSC Biblioteca Universitária. http://www.bu.ufsc.br/framerefer.html

BAUMER, C. Et alli. (s.d.) Grasshopper - An Universal Agent Platform Based on OMG MASIF and FIPA Standards. Berlin. http://www.grasshopper.de/links

BIESZCZAD, Andrzej et alli. (1998). Mobile Agents for Network Management. In: **IEEE Communications Surveys** - Fourth Quarter - v.1, n.1.

CASTILLO, Alberto et alli. (s.d.) **Concordia as Enabling Technology for Cooperative Information Gathering**. Mitsubishi Electric ITA, Waltham/EUA. http://www.concordiaagents.com/documents.htm

CHAVEZ, A. & Maes, P. (1996). Kasbah: An Agent Marketplace for Buying and Selling Goods. In: **Proceedings the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology** (PAAM-96), London, 22-24 april, p.75-90.

DI PIETRO, E. & La Corte, A. & Tomarchio, O. & Puliafito, A. (2000). Extending the MASIF Location Service in the MAP Agent System. In: **IEEE Symposium on Computer Communications** (ISCC2000), Antibes(France), july 2000, 6p.

FININ, T. & Wiederhold, G. (1991). **An Overview of KQML: A Knowledge Query and Manipulation Language**, Department of Computer Science, Stanford University.

FONER, Leonard N. (1997). Yenta: A Multi-Agent, Referral-Based Matchmaking System. Presented at **The First International Conference on Autonomous Agents (Agents'97),** Marina Del Rey, CA/EUA.

_____ (1996). A Multi-Agent Referral System for MatchMaking, In: **Proceedings the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96),** London, 22-24 april, p. 245-262.

FRANÇA, Júnia L. (1998). **Manual para Normalização de Publicações Técnico-Científicas.** 4 ed. Belo Horizonte. Editora UFMG.

GENESERETH, M. R. & Ketchpel, S. P. (1994). Software Agents, **Communications of the ACM** v.37, n.7, p.48-53.

GLASS, Graham (1998). **Voyager vs. Aglets**. Mensagem recebida por T.Papaioannou@lboro.ac.uk;voyager-interest@objectspace.com;aglets@kclink.com;mobility@media.mit.edu;dist-obj@cs.caltech.edu; em 10 setembro de 1998.

GUGLIELMELI, Evandro. (1995). **MOCO - Linguagem de Programação para Sistemas de Controle em Tempo Real**. Uberlândia, 115 p. Dissertação (Mestrado em Engenharia Elétrica) - Curso de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Uberlândia.

HARRISON, C. G. Et alli (1995). Mobile Agents: Are they a good idea ?, IBM Research Report, RC 19887.
http://www.research.ibm.com/xv-d953-mobag.ps.

HAYES-ROTH, B. (1995). An Architecture for Adaptative Intelligent Systems, **Artificial Intelligence**. V.72, n.1-2, p.329-365.

HUHNS, M. N. & Singh, M. P. (1994). **Distributed Artificial Intelligence for Information Systems**, CKBS-94. Tutorial. University of Keele/UK, june 15.

IBM Tokyo Research Laboratory (2000). **Aglets Workbench 1.1.0 beta 4.** Tokyo, 17-aug. http://sourceforge.net/projects/aglets/

_____ (2000). **Aglets Workbench 1.1 Beta 3**. Tokyo, 23-jun. http://www.trl.ibm.co.jp/aglets/

_____ (1998). **Aglets Workbench 1.0.3**. Tokyo, 21-aug. http://www.trl.ibm.co.jp/aglets/

IKV++ GmbH. (1999). Grasshopper - A Platform for Mobile Software Agents. Berlin. http://www.grasshopper.de/links

_____ (2000). **Grasshopper Programmer's Guide** - Release 2.1. Berlin. http://213.160.69.23/grasshopper-website/downexamples.html

_____ (2000). **Grasshopper Programmer's Guide** - Release 2.0. Berlin. http://213.160.69.23/grasshopper-website/downexamples.html

_____ (2000). **Grasshopper User's Guide** - Release 2.0. Berlin. http://213.160.69.23/grasshopper-website/downexamples.html

_____ (2000) Grasshopper 2.1 Core System and MASIF Addon - http://213.160.69.23/grasshopper-website/downcore.html

JOHANSEN, Dag. (1999). Mobile Agent Applications. **IEEE Concurrency**, p.80-82, Jul-Sept. Entrevista concedida a Dejan Milojicic.

KERN, Eduardo (1998). **Uma Estrutura de Agentes para o Processo de Licitação.** Florianópolis. 107 p. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina.

KOBLICK, Reuven (1999). Concordia. **Communications of the ACM** v.42, n.3, p.96-97.

KOTZ, Dave. (1999). Mobile Agent Applications. **IEEE Concurrency.** p.83-85, jul-sept. Entrevista concedida a Dejan Milojicic.

KOZIEROK, R. & Maes, P. (1993). A Learning Interface Agent for Scheduling Meetings. **Proceedings of the ACM-SIGCHI International Workshop on Intelligent User Interfaces**, Florida, p.81-93.

LA CORTE, A. & Puliafito, A. & Tomarchio, O. (1999). An Agent based framework for mobile users. In: **ERSADS'99**, Madeira (Portugal), 23-28 april, 4p.

LAKATOS, Eva M. (1992). **Metodologia do Trabalho Científico:** procedimentos básicos, pesquisa bibliográfica, projeto e relatório, publicações e trabalhos científicos. 4.ed. São Paulo: Atlas.

LANGE, Danny B. (1999). Mobile Agent Applications. **IEEE Concurrency.** p.85-86, jul-sept. Entrevista concedida a Dejan Milojicic.

_____ (1998) **Re: Voyager vs. Aglets**. Mensagem recebida por aglets@javalounge.com; voyager-interest@objectspace.com; mobility@media.mit.edu; dist-obj@cs.caltech.edu, em 13 setembro 1998.

_____ & ARIDOR, Y. (1997) **Agent Transfer Protocol - ATP/0.1.** IBM Tokyo Research Laboratory, march.

_____ & OSHIMA, M. (1998). **Programming and Deploying Java Mobile Agents with Aglets**. Reading: Addison Wesley Longman, 1998.

_____ & OSHIMA, M. (1999). Seven Good Reasons for Mobile Agents. **Communications of the ACM**, v. 42, n. 3, p. 88-91, march.

LAZAR, S. E Sidhu, D. (1997). **Discovery: A Mobile Agent Framework for Distributed Application Development** - Technical Report, Maryland Center for Telecommunications Research, University of Maryland Baltimore County.

LIEBERMAN, H. (1995). Letizia: An Agent that Assists Web Browsing, In **Proceedings of IJCAI 95**, AAAI Press.

MAF Cosubmission (1997). **The Mobile Agent System Interoperability Facility (MASIF/MAF)**. Presented by MAF Team OMG, 20/11/97.

MAF Team (1997). **The Mobile Agent Systems Interoperability Facility (MASIF/MAF) Presentation**, dezembro.

MAGNIN, L. & ALIKACEM El H. (1999). GenA : Multiplataform Genereic Agents. In: **First International Workshop on Mobile Agents for Telecommunication Applications** (MATA99). Ottawa, oct 6-8.

MILOJICIC, Dejan et alli. (1999). **Mobility: Processes, Computers, and Agents.** Reading: Addison-Wesley Pub. Co., 1999.

_____ (1998). **The OMG Mobile Agent System Interoperability Facility**. The OMG Mobile Agent System, p. 629-641.

MITSUBISHI Electric. (2001). **Concordia Version 1.1.7** (Evaluation Kit). Information Technology Center America, Waltham/EUA, 01-jan. http://www.concordiaagents.com/whatsnew.htm

_____ (1998). **Mobile Agent Computing**. Information Technology Center America, Waltham/EUA, 19-jan. http://www.concordiaagents.com/documents.htm

_____ MITSUBISHI Electric. (1998). **Tecnology at a Glance - Concordia - Java Mobile Agent Technology**. Information Technology Center America, Waltham/EUA, jan. http://www.concordiaagents.com/documents.htm

MÜLLER, Francisco J. (1994). **Extensões Funcionais para a Linguagem Forth.** Uberlândia. 109 p. Dissertação (Mestrado em Engenharia Elétrica) - Curso de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Uberlândia.

NWANA, Hyacinth S. (1996). *Software* Agents: An Overview. In: **Knowledge Engineering Review**. Cambridge University Press. V.11, n.3, p.1-40, sept 1996.

OMG (1998) MASIF: Mobile Agent Systems Interoperability Facility Specification, OMG TC Document ORBOS/1998-03-09, também disponível em: http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm

_____ (2000). **MASIF: Mobile Agent System Interoperability Facility**. Framingham / EUA, Jan. (full especification 2000-01-02), também disponível em: http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm

OSHIMA, M. Et alli (1998). **Aglets Specification 1.1 Draft.** IBM Tokyo Research Laboratory. 08-sept.

PAPAIOANNOU, Todd (1998). **Thoughts about MA systems**. Mensagem recebida por aglets@kclink.com; mobility@media.mit.edu; dist-obj@cs.caltech.edu em 10 setembro.

PERDIKEAS, M. K. Et alli (1999). Mobile agent standards and available platforms. **Computer Networks** (31), p.1999-2016.

PETRIE, Charles. (1999). Mobile Agent Applications. **IEEE Concurrency**, p.86-89, jul-sept. Entrevista concedida a Dejan Milojicic.

PULIAFITO, A. & Tomarchio, O. & Vita, L. (2000). **Design and Implementation of a Mobile Agents Platform**, (a ser publicado), 27p.

REZENDE, Marcos F. De. (1992). **Desenvolvimento de um Robô Móvel Autônomo Inteligente Utilizando a Arquitetura de Assunção.** Uberlândia. 102 p. Dissertação (Mestrado em Engenharia Elétrica) - Curso de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Uberlândia.

RHODES, B. J. & Starner, T. (1996). Remembrance Agent: A Continuously Automated Information Retrieval System. In: **Proceedings the First International Conference on the Practical Application of Intelligent Agents and Mullti-Agent Technology** (PAAM-96), London, 22-24 april, p.487-496.

ROESLER, Valter. (1999). **Redes Locais de Computadores.** São Leopoldo: PRAV/Unisinos, mar. 406 p. (mimeo).

RYGAARD, Chris. (1999). Mobile Agent Applications. **IEEE Concurrency**, p.89-90, jul-sept. Entrevista concedida a Dejan Milojicic.

SAMARAS, G. Et alli (1999). Mobile Agent Platforms for Web Databases: A Qualitative and Quantitative Assessment. In: **Procedings of Agent System and Applications / Mobile Systems** (ASAMA-99), Palm Springs, p.50-64.

SANTOS, Gildenir C. (2000). **Manual de Organização de Referências e Citações Bibliográficos para Documentos Impressos e Eletrônicos.** Campinas: Editores Associados; Editora da Unicamp.

SHARDANAND, U. & Maes, P. (1995), Social Information Filtering for Automating Word of Mouth, In **Proceedings of CHI-95**, Denver, CO., may.

SOARES, Luiz Fernando G. et alli. (1995). **Redes de Computadores** - Das LAN's, MAN's e WAN's às Redes ATM. 2.ed. Rio de Janeiro: Campus,1995.

TAI, H. & KOSAKA, K. (1999). The Aglets Project. **Communications of the ACM**, v.42, n.3, p.100-101, march.

TANENBAUM, Andrew S. (1997). **Redes de Computadores**. 3.ed. Rio de Janeiro: Campus, 1997.

TERPLAN, Kornel. (1999). **Web-Based Systems & Network Management - Advanced and Emerging Communications Technologies Series**. CRC Press.

VALDO, Clayton A. (2000). **CUBe: Um Sistema para Integrar Comércio Eletrônico e ERP através de Agentes Móveis**. Florianópolis. 98 p. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós Graduação em Ciência da Computação, Universidade Federal de Santa Catarina.

WALSH, T., PACIOREK, N., WONG, D. (1998) Security and Reliability in Concordia. In: **Procedings of the 31st Hawaii International Conference on Systems Sciences**, n.VII; p.44-53, jan.

ANEXO 1

PROTOCOLO ATP - EBNF (*Extended Backus-Naum Form*)

Terminologia

- Agente (*agent*) - Um programa que executa, com alguma autonomia, operações para um usuário.
- Conexão (*connection*) - Um circuito virtual da camada de transporte que se estabelece entre dois programas com a finalidade de comunicação.
- Mensagem (*message*) - Uma unidade básica de uma conexão ATP em um fluxo de bytes de acordo com uma sintaxe definida.
- Requisição (*request*) - Uma mensagem de requisição ATP.
- Resposta (*response*) - Uma mensagem de resposta ATP.
- Corpo da Mensagem (*message body*) - A parte "útil" em uma requisição ou resposta.
- Remetente (*sender*) - Um aplicativo ou sistema que estabelece uma conexão com a finalidade de enviar requisições.
- Destinatário (*recipient*) - Um aplicativo ou sistema que aceita uma conexão com a finalidade de receber requisições e enviar respostas.
- Serviço de agentes (*agent service*) - Um sistema capaz de agir tanto como um remetente quanto como destinatário de agentes.
- Proxy - Um sistema intermediário que age tanto como remetente quanto como destinatário com o propósito de fazer requisições um nome de usuários.

a) Versão do ATP

ATP_*version*="ATP/" *Major.Minor*

Major = *Digit*$^+$

Minor = *Digit*$^+$

b) Endereçamento URI

O endereçamento URI (*Uniform Resource Identifiers*) é uma *string* e representa a combinação de URL (*Uniform Resource Locator*) e URN (Uniform Resource Names).

ATP_URL = *Service_host* [*Agent_resource* | *Class_resource*]

Service_host = "atp://" *Host* [: *Port*]

Host = < Um domínio Internet ou endereço IP >

Port = *Digit*$^{+}$

Agent_resource = [*Name*] "#" *Agent_identifier*

Name = "/" < Uma *string* >

Class_resource = *Class_path*

Class_path = < Uma especificação de caminho válida >

Um exemplo de utilização:

Para serviços na IBM TRL: atp://joe.trl.ibm.com:434

Para um agente específico na IBM TRL, como parte de um conjunto de agentes de Danny: atp://joe.trl.ibm.com/danny#2874678383

c) Identificador de agente

Agent_identifier = < Uma *string* alfanumérica >

d) Formato de data ATP

ATP_*date* = < Sintaxe padrão IETF >

Por exemplo: "Sat, 12 Jan 2001 23:15:00 GMT"

e) Mensagem ATP

ATP_*message* = *Request* | *Response*

f) Request

Request = *Request_Line* **CRLF** *Header_field** **CRLF** [*Message_body*]

Request_line = *Method* **SP** *resource_URI* **SP** *ATP_version* **CRLF**

f.1) Method

Method = **DISPATCH** | **RETRACT** | **FETCH** | **MESSAGE** | *Extension_method*

Extension_method = *Token*

DISPATCH, RETRACT, FETCH E MESSAGE são os métodos para manuseio de agentes descritos anteriormente.

f.2) URI

Resource_URI = < URI relativo > | < URI absoluto >

O URI relativo depende da natureza da requisição (request) e da utilização ou não de *proxy*. O URI absoluto é utilizado quando o destinatário é um *proxy*.

Dispatch - ["/" *Name*]. Por exemplo: /joe

Retract - [*Name*]. Poe exemplo: /joe/2874678383 ou #2874678383

Fetch - [< Caminho do arquivo >]. Por exemplo: /agents/classes/Hello.class

Message - [*Name*] # *Agent_identifier*. Por exemplo: /joe#2874678383

g) *Response*

Response = *Status_line* **CRLF** *Header_field** **CRLF** [*Message_body*]

Status_line = *ATP_version* **SP** *Status_code* **SP** *Reason_phrase* **CRLF**

h) *Header* (Cabeçalho)

Header_field = *Name* : [*Value*] **CRLF**

Value = < Texto, exceto CR/LF >

A tabela 1, a seguir ilustra a utilização dos campos do cabeçalho para os métodos *request* e *response*. Alguns campos são relacionados ao destinatário, outros também ao remetente.

	Request				Response
	Dispatch	Retract	Fetch	Message	
Date	Sim	Sim	Sim	Sim	Sim
User-Agent	Sim	Sim	Sim	Sim	Sim
From	Sim	Sim	Sim	Sim	Não
Agent-System	Sim	Sim	Sim	Sim	Sim
Agent-Language	Sim	Não	Sim	Sim	Sim
Content-Type	Sim	Não	Não	Sim	Sim
Content-Encoding	Sim	Não	Não	Sim	Sim
Content-Lenght	Sim	Não	Não	Sim	Sim
Agent-Id	Sim	Sim	Não	Sim	Sim

Tabela 1 - Campos do cabeçalho utilizados por uma mensagem *dispatch*, *retract*, *fetch* ou *message* no protocolo ATP.

A seguir são apresentados uma breve descrição de cada campo, sua representação EBNF e um exemplo.

Date - Indica a data e hora em que a mensagem foi criada. Válida para o remetente e destinatário.

Date = "**Date:**" *ATP_date*

Ex: "Sat, 12 Jan 2001 23:15:01 GMT"

User-Agent - Contém informações sobre o aplicativo da qual a mensagem é originada.

User-Agent = "**User-Agent:**" *Comment*

Comment = < Descrição textual >

Ex: User-Agent: Tahiti aglet viewer

From - Contém o email do usuário humano responsável pelo aplicativo da qual a mensagem se origina.

From = "**From:**" *Mail_address*

Mail_address = < Endereço de correio eletrônico >

Agent-System - Contém informações sobre a plataforma da qual de originou o agente.

Agent_system = "**Agent-System:**" *Comment*

Comment = < Descrição textual >

Ex: Agent-System:ibm.aglets/Alpha4b

Agent-Language - Contém informação sobre a linguagem necessária à execução ou interpretação da mensagem (classe ou classes incluídas na mensagem).

Agent_language = "**Agent-Language:**" *Comment*

Comment = < Descrição textual >

Ex: Agent-Language:java

Content-Type - Usado para especificar o tipo de mídia da mensagem, podendo ser nulo.

Content_type = "**Content-Type:**" *Comment*

Comment = < Descrição textual >

Ex: Content-Type: x-aglets

Content-Encoding - Utilizado para indicar o mecanismo de codificação aplicado à mensagem, e por conseguinte o mecanismo de decodificação deve ser aplicado para se obter o conteúdo original da mensagem.

Content_encoding = "**Content-Encoding:**" *Comment*

Comment = < Descrição textual >

Ex: Content-Encoding: gzip

Content-Lenght - Indica o tamanho do corpo da mensagem a ser transferido, sem levar em conta o tipo e codificação.

Content_lenght = "**Content-Lenght:**" *Lenght*

Lenght = < Número decimal >

Ex: Content-Lenght: 8401

Agent-Id - Contém o identificador do agente, utilizado para referência ao agente para utilização em, por exemplo, especificar o agente a ser buscado (método *Retract*).

Agent-Id = "**Agent-Id:**" *Agent_identifier*

Agent_identifier = < Indicação de agente específico, em podendo ser numérico ou alfanumérico >

Ex: Agent-Id: 2874678384

i) Status Code e Reason Phrase

O *Status code* é um inteiro de 3 dígitos colocado pelo destinatário que indica o resultado da tentativa de processamento da mensagem. O *Reason phrase* é destinado a fornecer uma descrição textual do conteúdo do *status code*.

Status_code = $Digit^{+}$

Reason_phrase = < Texto sem incluir CR/LF >

O status code, pode ser entendido assim:

Iniciando em 1, indica que a mensagem foi recebida, entendida e corretamente processada. Iniciando em 2, indica necessidade de uma ação posterior que deva ser tomada. Iniciando em 3 indica algum tipo de erro. E finalmente iniciando em 4 indica a impossibilidade do destino de processar a mensagem. A tabela 2, a seguir, exemplifica alguns códigos de *status* mais comuns.

Status	Significado	Descrição
100	Ok	A requisição foi bem sucedida.
200	Moved	O recurso solicitado não está mais disponível.
300	Bad Request	O destinatário foi incapaz de entender a mensagem devido e erros de sintaxe.
301	Forbidden	O destinatário entendeu a mensagem mas não pode atendê-la, por alguma restrição imposta.
302	Not found	O destinatário não pode encontrar a recurso solicitado.
400	Internal recipient error	O destinatário encontrou uma condição inesperada que o esta impedindo de atender a requisição.
401	Not implemented	O destinatário não suporta a funcionalidade requerida para completar a requisição.
402	Bad gateway	O destinatário, agindo como um *proxy* ou *gateway*, não foi capaz de encontar o recurso solicitado.
403	Service Unavailable	O destinatário encontra-se incapaz de atender à requisição devido a sobrecarga com outras mensagens.

Tabela 2 - Valores de *Status Code* em uma mensagem de retorno.