Um Estudo sobre a Interoperabilidade em Ambientes Distribuídos

Thaís Vasconcelos Batista
thais@inf.puc-rio.br

Noemi de La Rocque Rodriguez
noemi@inf.puc-rio.br

Departamento de Informática - PUC-Rio
R. Marquês de São Vicente, 225 - Gávea
Rio de Janeiro - RJ
22453-900, Brasil

PUC-RioInf.MCC36/96 Outubro, 1996

Resumo

O desenvolvimento de aplicações distribuídas requer suporte para interação e integração entre programas executando em diferentes sites e em diferentes plataformas de software. Os mecanismos que promovem interoperabilidade em sistemas distribuídos incluem componentes de software, proposta usada por padrões como CORBA e OLE/COM, e agentes de software. Este trabalho analisa estas duas diferentes abordagens, apresentando os conceitos básicos e os relacionamentos entre eles. Algumas propostas específicas, que são consideradas representações das tendências atuais, são também discutidas.

Palavras-chave: Sistemas Distribuídos, Interoperabilidade, Agentes, Componentes, Orientação a Objetos, Paradigma Cliente-Servidor, Middleware, KQML, AgenTalk, Java, CORBA.

Abstract

Development of distributed applications requires support for interaction and integration between programs running on different network sites, frequently on different software platforms. Current mechanisms for the provision of interoperability in distributed systems include software components, as proposed by standards such as CORBA and OLE/COM, and software agents. This report studies these two different approaches, presenting basic concepts and interrelationships between them. Some of the specific proposals, which were considered representative of current trends, are also discussed.

Keywords: Distributed Systems, Interoperability, Agents, Components, Object Orientation, Client-Server Paradigm, Middleware, KQML, AgenTalk, Java, CORBA

Sumário

1. Introdução 1

2. Requisitos de Interoperabilidade em Ambientes Distribuídos 2

3. Componentes de Software 3

3.1 Origem da Idéia 4

3.2 Características e Propriedades de Componentes 5

3.3 Middleware 6

3.4 CORBA 8

4. Agentes 12

4.1 Definições de Agentes 12

4.2 Atributos Fundamentais de Agentes 12

4.3 Atributos Adicionais de Agentes 12

4.4 Caracterização de Agentes 13

4.4.1 Método de Ativação 13

4.4.2 Localização de Comportamento 13

4.4.3 Tipo de Conhecimento 15

4.4.4 Origem do Conhecimento 15

4.5 O Programa Agente 15

4.6 Comparação com outras abordagens 16

4.6.1. Programação Orientada a Agentes X Programação Orientada a Objetos 16

4.6.2. Agentes X Script 16

4.6.3. Computação Baseada em Agentes X Modelo Cliente-Servidor 16

4.7 Agentes Móveis 17

4.8. Análise de Trabalhos em Agentes de Software 18

4.8.1. Um Esquema baseado em Agentes para permitir interoperabilidade 19

4.8.2. KQML: Knowledge Query and Manipulation Language 23

4.8.3. AgenTalk 29

4.8.4. Obliq 32

4.8.5. Java 34

5. Conclusão 37

Bibliografia 39

1. Introdução

Atualmente, há uma crescente demanda por programas que possam interoperar - isto é - trocar informações e serviços com outros programas e até mesmo atuar em conjunto na resolução de problemas. Segundo Bernstein [4], interoperação entre programas significa que um programa em um sistema computacional pode acessar programas e dados em outro sistema e interpretá-los de maneira adequada.

O grande interesse por esta área nos dias atuais não acontece em um ambiente estático. Uma força enorme - simbolizada pelo interesse explosivo na Internet - está induzindo a ligação de computadores em redes, e é necessário uma estratégia comum que possa juntar tudo. Se duas máquinas estão fazendo qualquer colaboração sofisticada através desta rede, programadores precisam escrever programas especiais que garantam que tais máquinas estão prontas para interagir entre si [37]. Pesquisadores de diversas subáreas da ciência da computação estão preocupados em repensar as técnicas de programação usadas atualmente, de forma a permitir a interoperação de programas.

Um ponto crítico que torna difícil a interoperação entre programas é a heterogeneidade em ambientes de software [14]. Os ambientes se caracterizam por possuir uma variedade de sistemas de hardware heterogêneos, incluindo computadores pessoais, estações de trabalho, e mainframes. Estes sistemas executam sistemas operacionais diferentes e funcionam em diferentes arquiteturas de rede. Além disso, programas são escritos em linguagens diferentes e geralmente para serem executados em um sistema operacional específico, sem preocupação com questões de interoperação com outros programas. Consequentemente, em geral, estes programas oferecem interfaces diferentes. Como resultado, a integração é difícil.

Dentre as pesquisas que têm sido direcionadas para resolução do problema da interoperabilidade entre programas duas correntes estão se destacando, a Engenharia de Software baseada em Agentes e Tecnologia de Software Orientado a Componentes. A Figura 1 mostra uma taxonomia destas tecnologias para interoperabilidade.

A **Engenharia de Software baseada em agentes** [15] visa permitir a criação de software capaz de interoperar. Para isto, os programas de aplicação são escritos como **Agentes de Software**, isto é, componentes de software que se comunicam com seus pares trocando mensagens em uma linguagem de comunicação [14]. Essa linguagem deve permitir troca de dados e informação lógica, comandos individuais e scripts. Usando esta linguagem, agentes podem comunicar informações complexas e objetivos, além de poder programar uns aos outros de forma direta ou indireta. A Engenharia de Software baseada em Agentes tem sido apontada como um novo paradigma para resolução de problemas adequado para uma variedade de áreas [14].

A categoria de agentes é endereçada pelas linguagens que possam ser usadas para implementar agentes de software [24]. Uma linguagem de agente é aquela que permite programação de sistemas em termos de alguns dos conceitos desenvolvidos para agentes. Virtualmente qualquer linguagem de programação pode ser usada para desenvolvimento de agentes de software. Uma classe de linguagens que tem atraído muita atenção atualmente é a classe de linguagens de Script, especialmente aquelas designadas para programas móveis. Linguagens como Java, Tcl/Tk e Telescript fazem parte desta classe. Distingue-se esta classe de outra categoria, a de linguagens de comunicação. A categoria de linguagens de Comunicação são linguagens especificamente projetadas para descrever e facilitar a comunicação entre dois ou mais agentes. Esta classe está dissociada da classe de linguagens de script porque as linguagens de script em geral são designadas para controle de processos em uma única plataforma. Apenas algumas linguagens desta classe possuem primitivas de comunicação para desenvolvimento de agentes que incluem o transporte de um agente de uma máquina para outra.

A outra principal classe de tecnologia para interoperabilidade é a de **Componentes de Software**. Componentes são pedaços de software autônomos que podem trabalhar em redes, aplicações, linguagens, ferramentas e sistemas operacionais diferentes. A esta classe pertencem padrões e protocolos de coordenação como CORBA, OpenDoc, OLE, etc. O principal compromisso destas tecnologias é garantir que aplicações podem trocar estruturas de dados e métodos entre plataformas distintas. Em várias

publicações são empregados outros termos como sinônimos de componentes de software: objetos interoperáveis, objetos componentes, objetos distribuídos, componentware e middleware. Alguns destes termos referem-se apenas a um dos aspectos da tecnologia. Por exemplo, middleware refere-se apenas aos componentes de nível médio do sistema, ou seja, a camada de abstração acima do sistema operacional e abaixo das aplicações específicas [4] [36].

O objetivo neste trabalho é analisar as tecnologias que apresentam soluções para promover a interoperabilidade de software em ambientes distribuídos. Infelizmente não há uma terminologia padrão para muitos dos conceitos discutidos neste trabalho. Serão seguidos os termos mais comuns encontrados na literatura pesquisada.

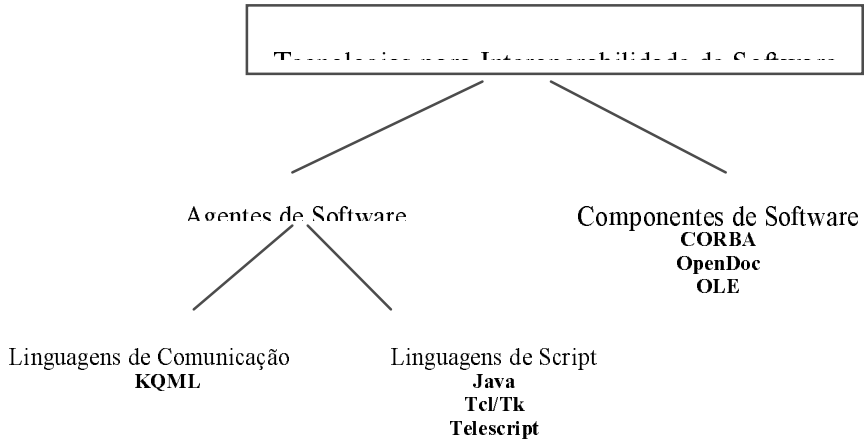


Figura 1 - Classificação de Tecnologias para Interoperabilidade

O trabalho está organizado da seguinte forma. Na seção 2 são definidos os requisitos para interoperabilidade em ambientes distribuídos. A seção 3 é destinada a considerações sobre componentes de software. Esta seção discute vários aspectos relacionados ao assunto e apresenta o *Common Object Request Broker Arquitecture (CORBA)*, um padrão para interoperabilidade de Sistemas Distribuídos que usa a tecnologia de componentes de software. A seção 4 tem por objetivo discutir a filosofia de agentes de software. Para isto, oferece uma introdução aos principais conceitos e características de agentes, compara-o com outras abordagens e analisa aspectos de interoperabilidade em alguns sistemas de agentes. A seção 5 finaliza o trabalho com uma análise crítica sobre as duas abordagens apresentadas como solução para o problema em questão.

2. Requisitos de Interoperabilidade em Ambientes Distribuídos

Como a finalidade neste trabalho é analisar as tecnologias que apresentam soluções para promover a interoperabilidade de software em ambientes distribuídos, é necessário enumerar os requisitos necessários para atingir a desejável interoperabilidade.

Diferentes graus de interoperabilidade podem ser tratados por um sistema:

- interoperabilidade entre objetos da mesma linguagem em ambientes operacionais diferentes. Neste contexto objetos de uma mesma linguagem estão distribuídos em uma rede, e a interoperabilidade pode ser oferecida a nível da linguagem de programação.

- interoperabilidade entre objetos de linguagens diferentes em um mesmo ambiente operacional. Neste caso os objetos não estão distribuídos ao longo de uma rede heterogênea,

então, para permitir a integração, é necessário um suporte de interoperabilidade entre linguagens.

- interoperabilidade entre objetos de linguagens diferentes e em ambientes diferentes. Este é o sentido mais amplo da interoperabilidade, através da qual todos sistemas podem ser interoperáveis. Este ponto deve ser endereçado por uma abordagem que ofereça a ilusão de um sistema único composto por uma variedade de máquinas híbridas.

A interoperabilidade no contexto deste trabalho será estudada no sentido mais amplo possível, ou seja, serão analisados as soluções oferecidas por diferentes sistemas que se enquadram nos diversos graus de interoperabilidade relacionadas acima, porém sempre procurando o foco no máximo grau de interoperabilidade oferecido pelo sistema. Dependendo do grau de interoperabilidade que o sistema se propõe a oferecer, alguns itens relacionados abaixo irão estar presentes ou não.

Segundo [17], para possibilitar a interoperabilidade entre objetos de diferentes linguagens em um ambiente distribuído, os seguintes aspectos devem ser endereçados:

R1 Linguagem padrão para permitir interoperação entre objetos escritos em diferentes linguagens.

R2 Localização de objetos em qualquer lugar da rede de maneira transparente para o usuário da aplicação. Aspectos de distribuição devem ser mascarados.

R3 Possibilidade de integração de diferentes objetos em uma aplicação.

R4 Abstração de serviços do sistema operacional. Isto significa que o suporte a interoperabilidade deve ser oferecido em um nível acima do sistema operacional para permitir interoperabilidade entre objetos em sistemas operacionais distintos.

R5 Abstração da plataforma de hardware de forma a permitir interoperabilidade entre plataformas de hardware distintas.

R6 Comunicação com qualquer recurso de outro sistema sem preocupacão com a pilha de protocolos que suporta esta comunicação.

R7 A solução para interoperabilidade deve ser aberta, isto é, qualquer parte da solução deve ser mutável.

3. Componentes de Software

A tecnologia de componentes de software se propõe a oferecer uma nova abordagem para se desenvolver aplicações não monolíticas. Aplicações monolíticas são aquelas construídas como um todo, incorporando uma gama de funções necessárias para satisfazer todas as possíveis necessidades dos seus diversos usuários. Esta filosofia de incorporar toda possível funcionalidade no produto, tem produzido softwares imensos e complexos, com sérias dificuldades de gerenciamento e cujo conjunto de funcionalidades raramente é totalmente utilizado por um único usuário. Além disso, a interoperabilidade entre aplicações monolíticas é extremamente difícil principalmente porque não há preocupação com este aspecto no projeto destas aplicações.

É necessário então modificar a maneira de se construir sistemas, visando produzir sistemas menos complexos e com a funcionalidade limitada à necessidade básica da maioria dos usuários, mas também que seja possível a incorporação rápida de novos módulos com funcionalidades avançadas diferentes para cada tipo de usuário.

A idéia é que componentes sejam pedaços de software autônomos que possam trabalhar entre redes, aplicações, linguagens, ferramentas e sistemas operacionais distintos. A finalidade com isto é permitir interoperação entre componentes escritos em linguagens diferentes e executando em plataformas diferentes. Esta filosofia na prática viabilizaria que aplicações monolíticas sejam substituídas por

aplicações formadas por um conjunto de componentes que serão interligados usando scripts. Desta forma, as complexas aplicações monolíticas atuais, que visam atender a todas as classes de usuários, poderão ser substituídas por aplicações personalizadas projetadas usando os componentes específicos para atender às necessidades do seu usuário particular. Caso tenha a necessidade de funções adicionais, o usuário deverá poder comprá-las na forma de componentes. Esta abordagem visa aumentar a reusabilidade de software, a independência da plataforma, e oferecer mais flexibilidade e liberdade de escolha.

A computação com componentes é também chamada computação com objetos distribuídos ou computação cliente-servidor orientada a objetos [12].

Nesta seção serão discutidos aspectos da tecnologia de componentes de software. Na subseção 3.1 é apresentada a origem da idéia de componentes. A subseção 3.2 identifica suas características e propriedades. A subseção 3.3 é dedicada à discussão sobre Middleware, uma classe especial de componentes que exerce uma importante função no processo de interoperação em ambientes distribuídos. Para finalizar, a subseção 3.4 discute aspectos gerais do padrão CORBA que visa permitir a interoperabilidade entre aplicações que o segue. Este padrão foi escolhido para maiores considerações no presente trabalho devido à sua crescente influência nas pesquisas no campo da interoperabilidade de software através de objetos.

3.1 Origem da Idéia

A computação com objetos distribuídos é resultado da convergência das tecnologias de orientação a objetos e cliente-servidor [12], onde são combinadas as vantagens da tecnologia cliente-servidor com a riqueza dos modelos orientados a objetos.

Da tecnologia de orientação a objetos foi herdada a organização do sistema em termos de objetos, bem como as propriedades dos objetos. Objetos oferecem um paradigma organizacional para decompor aplicações grandes em objetos cooperantes, bem como o paradigma do reuso para compor aplicações de objetos pré-empacotados. Objetos apenas conhecem os serviços oferecidos por outros objetos (suas interfaces), sem conhecer como eles oferecem tais serviços (sua implementação). Esconder detalhes da implementação dentro de objetos é uma das contribuições chave do paradigma de orientação a objetos para gerenciamento da complexidade na programação distribuída.

Apesar das muitas contribuições oferecidas pela tecnologia de orientação a objetos, há vários problemas abertos que devem ser resolvidos para atingir o objetivo de desenvolvimento de software orientado a componente: [30]

1 - mecanismos de orientação a objetos para composição e reuso devem ser integrados com outras características como concorrência, persistência e distribuição.

2 - o reuso efetivo de software propõe a existência de ferramentas para dar suporte a organização e recuperação de componentes de acordo com os requisitos da aplicação e a construção interativa de aplicações com componentes.

3 - como o design de esquemas reusáveis é um processo interativo e evolucionário, é necessário gerenciar software de tal maneira que o design e implementação sejam uma tarefa agradável.

Para resolver a questão da distribuição dos objetos foi adotado o modelo cliente-servidor por ser considerado o modelo padrão para estruturação de aplicações distribuídas. Neste modelo, as aplicações são estruturadas em processos que oferecem serviços (servidores) e processos que solicitam os serviços (clientes).

Combinadas a distribuição oferecida pelo modelo cliente-servidor com a reusabilidade de objetos através da publicação de sua interface, surgiu a tecnologia de componentes de software.

Em suma, computação com componentes de software ou computação com objetos distribuídos é uma extensão da tecnologia cliente-servidor. Contudo, há uma diferença no seu processo de trabalho e na sua implementação. Com cliente-servidor há geralmente uma aplicação executando em um computador cliente enquanto outra aplicação executa no computador servidor. Estas duas aplicações se comunicam via uma rede, usualmente através de algum serviço de comunicação oferecido na forma de uma API ou funções de biblioteca. Ou seja, cliente-servidor é uma versão restritiva da computação com objetos distribuídos. Uma aplicação distribuída é feita de objetos como qualquer outra aplicação cliente-servidor, porém, os objetos de uma aplicação com objetos distribuídos podem executar em múltiplos computadores ligados em uma rede.

Computação com objetos distribuídos é um ambiente de computação muito complexo e requer uma infra-estrutura robusta e sofisticada.

3.2 Características e Propriedades de Componentes

Como a idéia de componentes de software é a interoperação entre componentes escritos em linguagens diferentes e executando em plataformas diferentes, em termos gerais, dois aspectos oferecem o suporte à esta heterogeneidade e distribuição. O primeiro deles é uso de **interfaces de programação padrão**. A interface padrão possibilita o trabalho conjunto de componentes, mesmo se são implementados em linguagens diferentes. Além disso, o uso de um interface padrão facilita a portabilidade dos componentes para diversas plataformas que ofereçam suporte a esta interface. Assim, a interface padrão confere ao componente a independência de linguagens e de plataformas. O segundo aspecto que possibilita a interoperabilidade através de componentes é o uso de **protocolos padrão.** O uso de protocolos padrão permite que aplicações em plataformas diferentes interpretem as mensagens de maneira adequada.

Na literatura, não existe ainda um consenso sobre as propriedades de componentes de software. A seguir serão definidas as propriedades mínimas que um componente deve apresentar segundo [32]:

- **ser um objeto estendido:** um componente é um objeto no sentido de que dá suporte a encapsulação, herança e polimorfismo. É estendido porque também deve dar suporte a todas as características associadas a um objeto autônomo distribuído.

- **possuir uma interface bem especificada**: como um objeto, um componente pode apenas ser manipulado através de sua interface.

- **ser um objeto interoperável**: pode ser chamado como um objeto entre espaços de endereçamento, redes, linguagens, sistemas operacionais e ferramentas distintas. É uma entidade de software independente do sistema.

- **não ser uma aplicação completa**: ele pode ser combinado com outros componentes para formar uma aplicação completa. Ele é projetado para realizar um conjunto de tarefas limitadas dentro do domínio de uma aplicação.

- **poder ser usado em combinações imprevisíveis**: como objetos do mundo real, um componente pode ser usado de maneiras totalmente não previstas pelo desenvolvedor original.

Analisando as características dos componentes de software percebe-se uma estreita identificação com agentes de software. As propriedades acima comentadas que caracterizam um componente, caracterizam também a infra-estrutura de agentes. Na verdade, pode-se classificar agentes como componentes que fazem mais que interoperar; também cooperam com outros agentes em um nível semântico, podendo oferecer informações sobre si próprio e fazer inferências sobre seu comportamento.

Na junção dos componentes que formam as aplicações, os programadores terão de escrever código para interação entre os diversos componentes. Juntar componentes que não têm nenhuma ligação prévia é uma tarefa difícil, uma vez os programadores terão que entender o funcionamento de tais componentes,

adaptar as suas interfaces para que elas possam interagir e muitas vezes modificar o código de cada um deles. Neste cenário surge a necessidade de padrões que estabeleçam as regras de junção para a interação entre componentes diferentes, ou seja, a infra-estrutura necessária para se trabalhar com componentes distribuídos.

O OMG (Object Management Group) é um consórcio entre várias empresas que, desde 1989, tem trabalhado na especificação de uma arquitetura para um barramento de software aberto no qual objetos componentes escritos por diferentes vendedores possam interoperar. O barramento aberto para componentes distribuídos do OMG chama-se CORBA (Common Object Request Broker Architecture). O barramento de objetos do CORBA - Object Request Broker (ORB) - define a forma dos componentes que podem utilizá-lo e como estes componentes podem interoperar. O ORB é o elemento que faz o elo entre os objetos distribuídos. Ele permite aos objetos localizarem e ativarem outros objetos na rede, independentemente do processador ou linguagem de programação utilizada para desenvolver qualquer objeto. A exceção notável no OMG é a MicroSoft que tem seu próprio barramento de componentes chamado COM (Component Object Model), também conhecido como Network OLE.

3.3 Middleware

Para ajudar os clientes a resolverem problemas de heterogeneidade e distribuição, e assim habilitar a implementação de utilitários, estão sendo oferecidos serviços de sistemas distribuídos que têm protocolos e interfaces padrão. Estes serviços são chamados serviços de middleware porque eles se situam "no meio", em uma camada acima do sistema operacional e do software de rede e abaixo das aplicações específicas. A Figura 2 mostra o diagrama de um sistema com serviços de Middleware.

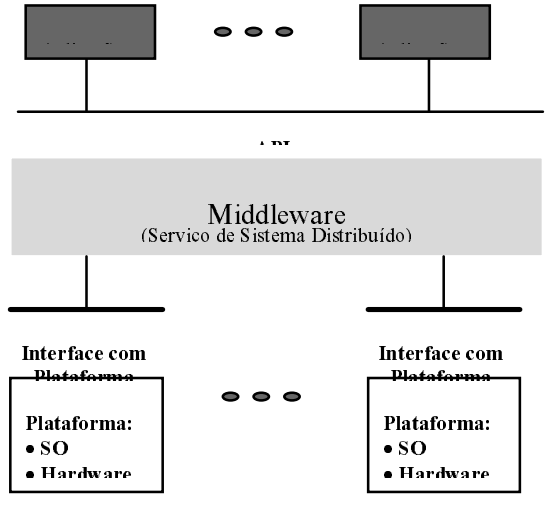


Figura 2 – Middleware

O middleware possibilita aos programadores de aplicações trabalharem com interfaces de alto nível que mascaram a complexidade das redes e dos protocolos. Por isto, para muitas aplicações, o middleware está se tornando mais importante que o sistema operacional e os serviços de rede dos quais elas eram dependentes. Por exemplo [ ], novas aplicacões geralmente dependem de um mecanismo de chamada remota de procedimento (RPC) [10] em vez de um mecanismo de mensagem a nível de protocolo da camada de transporte do modelo OSI [3]. Em geral, o middleware está substituindo as funções não distribuídas do sistema operacional por funções distribuídas que usam a rede (acesso a arquivos remotos, chamada remota de procedimento, etc).

Middleware é difícil de definir de maneira precisa, porém, componentes middleware têm várias propriedades que juntas tornam claro que o componente não é uma aplicação ou serviço específico de

plataforma. Para ser considerado um middleware, um componente deve apresentar as seguintes características:

- **ser genérico**

- **executar em múltiplas plataformas**

- **ser distribuído:** esta característica permite que o middleware seja acessado remotamente e que outros serviços e aplicações sejam acessados remotamente. Com isto, ele também aumenta a interoperabilidade, pois aplicações em diferentes plataformas podem usar o serviço para se comunicar e trocar dados. Um serviço de middleware acessado remotamente inclui:

    - uma parte cliente que usa as APIs de serviço executando no espaço de endereçamento da aplicação;

    - uma parte servidor que oferece as funções principais do serviço e pode executar em diferentes espaços de endereçamento, isto é, em diferentes sistemas.

- **seguir interfaces padrão:** para que aplicações que usem uma API existente possam usar novos serviços sem modificação da API, o middleware deve oferecer uma API padrão para todos os seus serviços.

- **seguir protocolos padrão:** por questões de interoperabilidade, o middleware deve seguir protocolos padrão (TCP/IP, OSI) ou no mínimo publicar o protocolo proprietário utilizado, como a IBM faz com o SNA [26]. Desta forma, múltiplas implementações do middleware podem ser desenvolvidas e estas implementações poderão interoperar.

Devido a importância de interfaces padrão para portabilidade da aplicação e protocolos padrão para interoperabilidade, o middleware tem sido sujeito de muitos esforços de padronização, alguns através de padrões formais como ISO e ANSI e outros através de consórcio da indústria de informática como X/Open, OSF e OMG.

Para exemplificação deste conceito serão listados a seguir serviços que possuem características que podem levá-los a tornarem serviços de middleware. Nem todos eles atualmente são distribuídos e portáveis, mas em grande parte deles estão sendo investidos esforços para incorporação destas características. São eles [ ]:

- **Gerenciamento de Apresentação**: gerenciador de impressão, gerenciador de gráficos, gerenciador de formulários.

- **Computação**: *sorting*, conversores de dados

- **Gerenciamento de Informação**: servidor de diretório, gerenciador de arquivos, sistema de banco de dados orientado a objetos

- **Comunicações**: Chamada Remota de Procedimento, Correio Eletrônico

- **Controle**: Gerenciador de *Threads*, Gerenciador de Transações

- **Gerenciamento do Sistema**: Serviço de Notificação de Eventos, Gerenciador de Configuração, Detector de Falhas, Serviço de Autenticação, Serviço de Criptografia.

Segundo Bernstein [ ], os componentes específicos que constituem um middleware deverão sofrer mudanças com sua evolução. Alguns serviços que estão atualmente no sistema operacional deverão migrar para se tornar um serviço de middleware e vice-versa. Este fato se deve à imaturidade atual do estado da arte onde existem muitas propostas mas nenhum padrão. O POSIX (*Portable Operating System Interface for Computing Evironment*) [23] é uma tentativa de padronizar a interface entre os

programas de aplicação e os serviços do sistema operacional. Vale ressaltar que o POSIX não estabelece normas de como devem ser escritos programas de aplicação ou sistemas operacionais, mas define uma maneira padrão para as aplicações obterem serviços básicos de um sistema operacional.

3.4 CORBA

Desde 1990, o OMG publica a sua Arquitetura de Gerenciamento de Objetos (CORBA), que consiste dos quatro elementos seguintes, representados na figura 3.

1) *Object Request Broker (ORB)* é o barramento de objetos do CORBA. Este aspecto será discutido com maiores detalhes ainda nesta seção.

2) *Serviços Comuns de Objetos:* oferecem interfaces de funções fundamentais para a construção de aplicações orientadas a objetos distribuídas. Definem conjuntos de objetos que realizam funções fundamentais como nomes, eventos, persistência, relacionamento, segurança, concorrência, etc.

3) *Facilidades Comuns:* oferece interfaces de funções comumente utilizadas pelas aplicações. O OMG ainda está trabalhando na especificação dos serviços desta camada. As facilidades comuns foram divididas em duas categorias: Facilidades Horizontais e Verticais. As facilidades horizontais incluem funções necessárias a maioria dos sistemas independentemente do propósito a que eles se destinam. Incluem serviços de interfaces com usuário, de infra-estrutura para automação de tarefas (gerenciamento de tarefas), etc. As facilidades verticais incluem funções para uso por aplicações de domínios específicos como sistemas CAD, sistemas bancários, etc.

)*Objetos de Aplicação:* são objetos definidos pelo usuário do CORBA.

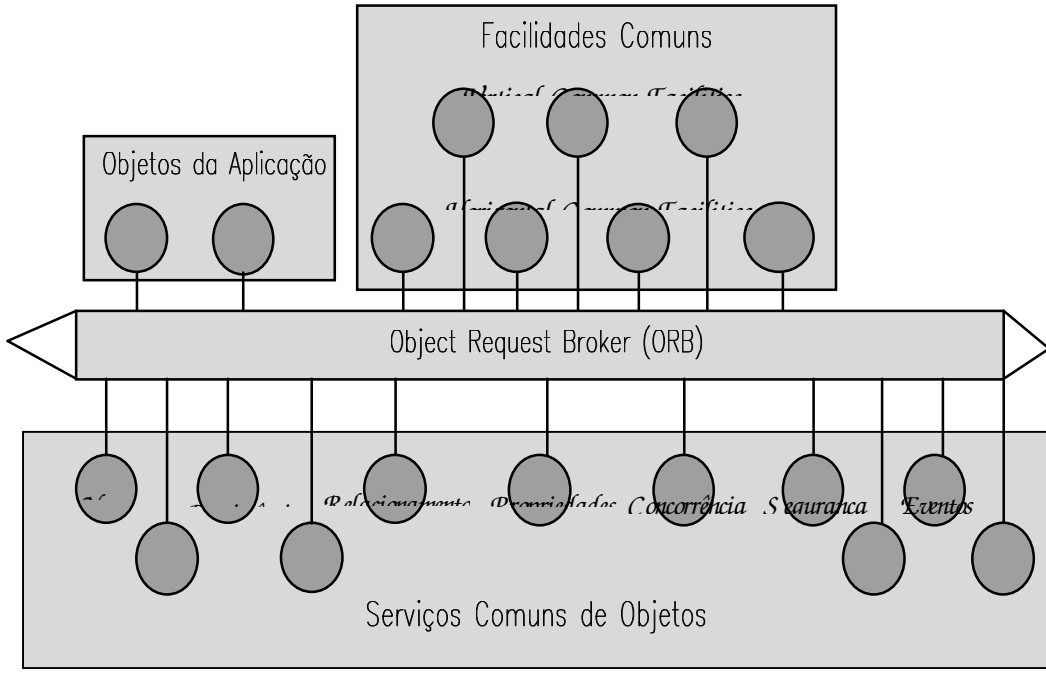


Figura 3 - Arquitetura do CORBA

Como este trabalho visa comentar o suporte à interoperabilidade oferecida pelo CORBA, não serão comentados todos os elementos da sua arquitetura, mas apenas aqueles diretamente envolvidos no suporte à interoperabilidade.

O ORB é o barramento de objetos que permite aos objetos fazerem solicitações e receberem respostas de outros objetos localizados localmente ou remotamente. Os mecanismos usados para se comunicar, ativar ou armazenar os objetos no servidor não precisam ser conhecidos pelo cliente, ou seja, são transparentes para o desenvolvedor. O ORB é então, o elemento integrador dos objetos distribuídos, o middleware da computação com objetos distribuídos que permite interoperabilidade em redes de objetos heterogêneos.

A flexibilidade oferecida pelo ORB para o trabalho com objetos distribuídos provém dos seguintes aspectos oferecidos por ele:

- o ORB permite chamada estática ou dinâmica de métodos, isto é, as chamadas de métodos podem ser definidas em tempo de compilação ou em tempo de execução. O usuário pode optar por usar a checagem de tipo em tempo de compilação ou a flexibilidade associada com a ligação dinâmica;

- o ORB permite chamar métodos em objetos servidores usando a linguagem de alto nível de preferência - atualmente C, C++ ou Smalltalk - independentemente da linguagem na qual os objetos são escritos. Como veremos mais adiante, isto é possível devido à separação entre interface e implementação dos objetos e devido à definição de tipos de dados neutros para linguagens usando a linguagem IDL, o que torna possível chamar objetos de várias linguagens e diferentes sistemas operacionais;

- cada ORB possui um **Repositório de Interfaces** que contém a descrição das funções que os servidores oferecem e seus parâmetros. Os cliente usam metadados para descobrir como chamar serviços em tempo de execução.

Em suma, o ORB é o middleware que estabelece relacionamento cliente-servidor entre os objetos. Através de um ORB um cliente pode transparentemente chamar um método em um objeto servidor que pode estar na mesma máquina ou em qualquer lugar da rede. O ORB intercepta a chamada e é responsável por encontrar um objeto que possa implementar a solicitação, passar os parâmetros, chamar o método e retornar os resultados. O cliente não precisa tomar conhecimento de onde o objeto está localizado, da linguagem de programação na qual ele foi implementado, do Sistema Operacional onde ele executa ou de qualquer outro aspecto do sistema que não seja parte da interface do objeto. Desta forma, o ORB oferece interoperabilidade entre aplicações em máquinas diferentes em um ambiente distribuído heterogêneo e interconecta múltiplos sistemas de objetos.

O objetivo do OMG é criar uma infra-estrutura de objetos verdadeiramente aberta. Através desta infra-estrutura o programador pode especificar a interface e a estrutura de objetos em uma linguagem padrão (IDL). Objetos distribuídos podem se comunicar através desta interface padrão. O Repositório de Interfaces, que contém as definições de todas as interfaces, permite componentes descobrirem os outros dinamicamente em tempo de execução. Sistemas clientes/servidores complexos podem ser unidos juntando-se componentes e estendendo-os. Objetos podem ser modificados sem afetar o restante dos componentes do sistema ou a forma como eles interagem. Desta forma, uma aplicação cliente/servidor consiste de uma coleção de componentes em colaboração.

O CORBA cria especificações de interface, e não código. As especificações são escritas em uma Linguagem para Definição de Interface (IDL) que define as interfaces de um componente. A IDL é puramente declarativa, o que significa que ela não oferece detalhes de implementação. A IDL pode ser usada para especificar os atributos de um componente, as suas superclasses, os tipos de eventos que ele emite e os métodos da sua interface, incluindo os parâmetros de entrada e de saída e seus tipos de dados. Como IDL é uma linguagem para descrever interfaces, componentes escritos em IDL devem ser portáveis entre linguagens, ferramentas, sistemas operacionais e redes; além disso, estes componentes devem ser capazes de interoperar entre barramentos de objetos CORBA de múltiplos fabricantes. A gramática IDL é um subconjunto de C++ com palavras-chave adicionais para dar suporte a conceitos distribuídos. Os métodos especificados em IDL podem ser escritos e chamados de qualquer linguagem que ofereça amarrações CORBA (CORBA *bindings*) - atualmente C, C++ e Smalltalk (novas versões de

Ada, COBOL e Objective C que definem amarrações CORBA estão sendo desenvolvidas). Os programadores tratam com objetos CORBA usando construtores nativos da linguagem. A IDL oferece interfaces independentes de linguagens de programação e sistemas operacionais para todos os serviços e componentes que residem em um barramento CORBA. Ela permite objetos dos clientes e dos servidores escritos em diferentes linguagens interoperarem entre redes e sistemas operacionais (veja figura). Ou seja, a IDL é usada para definir sintaticamente as interfaces para objetos, para permitir invocação de operações em objetos com diferentes implementações. A partir das definições em IDL é possível mapear objetos CORBA em linguagens de programação específicas.

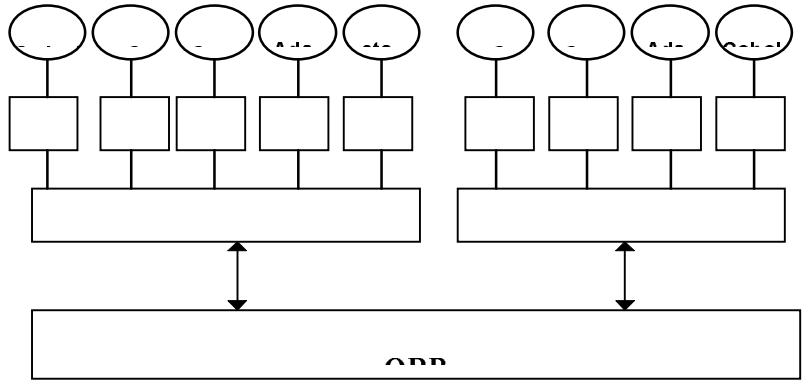


Figura 4 – Amarrações para a Linguagem IDL

O CORBA foi designado para permitir componentes inteligentes descobrirem os outros componentes e interoperarem através do barramento de objetos. Contudo, o CORBA vai além da interoperabilidade apenas. Ele também especifica um conjunto extenso de serviços relacionados ao barramento para criar e eliminar objetos, acessando-os por nome, armazenando-os em memórias persistentes, externalizando seus estados e definindo relacionamentos *ad-hoc* entre eles. Na arquitetura do CORBA, estes serviços estão localizados na camada *Common Objects Services*.

Objetos CORBA são pedaços que vivem em qualquer lugar da rede. Eles são empacotados como componentes binários que clientes remotos podem acessar através de métodos. A linguagem e o compilador usado para criar objetos são totalmente transparentes aos clientes. Os clientes não precisam saber onde os objetos distribuídos residem ou em qual sistema operacional executam. Pode ser no mesmo processo ou em uma máquina que se situa em qualquer ponto da rede. Além disso, os clientes não precisam saber como o servidor de objetos é implementado. P. ex., um servidor de objetos pode ser implementado como um conj. de classes C++ ou pode ser implementado com milhões de linhas de código COBOL - o cliente não sabe a diferença. O que o cliente precisa saber é a interface que o servidor de objetos publica. Esta interface serve como ligação entre os clientes e os servidores.

A especificação do CORBA 1.1 (1991) [???] apenas especifica a linguagem usada para definição da interface dos componentes, as ligações da linguagem e as APIs que possibilitam fazer interface com o ORB. De acordo com esta especificação, pode-se escrever programas portáveis a serem executados nos ORBs disponíveis. O CORBA 2.0 [31] especifica a interoperabilidade entre ORBs de diferentes fabricantes através do protocolo inter-ORB chamado Protocolo Geral Inter-ORB (GIOP). O Internet Inter-ORB (IIOP) é uma das variantes do GIOP. Todas as variantes do GIOP, como o IIOP, compartilham especificações comuns para [38]:

- representação comum de dados (CDR) incluindo convenções de empacotamento (*marshalling*)
- referência a objetos interoperáveis (IOR). Versões anteriores à versão CORBA 2.0 não especificam como uma referência a um objeto deve ser implementada pelos ORBs. Isto dificulta a interoperabilidade pois cada ORB pode implementá-la da forma que lhe convier. Para que seja

possível um objeto fazer uma solicitação a outro objeto que está em outro ORB, o ORB destino deve entender a referência passada pelo ORB origem. O IOR especifica a estrutura que uma referência a um objeto externo deve possuir. O IOR não precisa ser usado internamente em um ORB, mas apenas quando vai ser feita uma referência a um objeto em um ORB diferente.

- tipos interoperáveis

- conteúdo, formato e semântica da mensagem IOP.

Como ilustração da potencialidade do modelo CORBA, pode-se fazer uma comparação do CORBA com outras abordagens que dão suporte a objetos distribuídos como, por exemplo, Chamada Remota de Procedimento (RPC).

Comparando as chamadas de métodos no ORB e na Chamada Remota de Procedimento (RPC), identifica-se que os mecanismos são bastante similares, mas existem diferenças importantes. Em RPC, chama-se uma função específica, os dados são enviados como parâmetros. No ORB, chama-se um método dentro de um objeto específico. Diferentes classes de objetos podem responder a mesma chamada de método de forma diferente, devido ao polimorfismo característico dos objetos. Desta forma, cada objeto gerencia sua própria instância de dados privada, sendo o método implementado em uma instância específica de dados. A figura 5 ilustra o funcionamento. O RPC, por sua vez, não oferece o polimorfismo característico dos objetos, mas permite diferentes implementações de um programa atribuindo um diferente número de versão para cada implementação do programa. A diferença é que no ORB, devido ao inerente polimorfismo, a instanciação do método adequado é feita automaticamente, dependendo do objeto que está sendo referenciado, enquanto que no RPC, o programador tem de explicitamente endereçar a versão desejada.

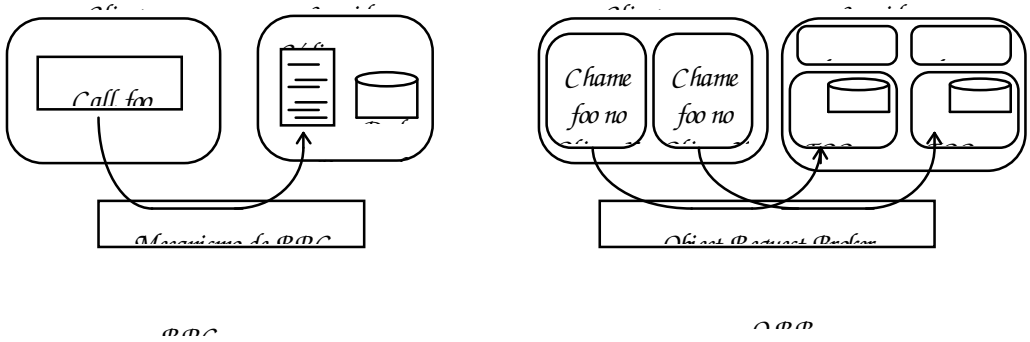


Figura 5 - ORB x RPC

No ORB, a chamada de métodos é feita para um objeto específico que controla dados específicos e implementa a função na sua classe específica. Em contraste, as chamadas RPC não são específicas, todas as versões de funções que têm o mesmo nome têm os mesmos tipos de parâmetros, apenas a implementação pode diferir de uma versão para outra.

Um aspecto interessante do CORBA é o suporte a referências a objetos que são resolvidas pelo ORB, o que proporciona a transparência de localização e transparência de migração. Este é um problema importante para sistemas que trabalham com objetos distribuídos. Por exemplo, no WWW são usadas URLs (Uniform Resource Locators) como *handles* para acessar recursos. URLs são associados com um recurso particular em uma localização particular, tornando esta forma de acesso dependente da localização do objeto. Então, se há uma migração do objeto ou do recuso, a localização com a URL antiga falha. Em contraste, o CORBA (versão 2.0) não apresenta este problema pois referências são resolvidas pelo ORB e ligadas dinamicamente.

A questão de transparência de acessos locais ou remotos oferecida pelo CORBA é questionável, uma vez que nenhuma novidade foi apresentada para resolver o problema do tempo de acesso e da confiabilidade. Não há como esconder o fato de que o acesso remoto é mais lento que o acesso local. Além disso, o inerente potencial para falha envolvido com acessos remotos é resolvido pelo CORBA através da inserção de tratamento de exceções.

4. Agentes

Nesta seção serão estudadas aspectos gerais da tecnologia de agentes de software.

Na subseção .1 são apresentadas definições para o termo agente. As subseções .2 e .3 se dedicam a definir, respectivamente, os atributos fundamentais e adicionais de um agente de software. A subseção . define uma série de características de sistemas agentes. Na subseção .5 são feitas considerações sobre um programa agente. A seguir, na subseção .6 é feita uma comparação de agentes com outras abordagens. Na subseção .7 discute-se uma classe especial de agentes, os agentes móveis, que são utilizados em aplicações cliente-servidor. Ainda nesta subseção são analisadas as vantagens do uso de agentes móveis como base dos serviços e aplicações cliente-servidor. Na subseção .8 é feita uma análise dos principais sistemas baseados em agentes presentes na literatura. Esta análise é conduzida à luz das características pertinentes à interoperabilidade

4.1 Definições de Agentes

"uma entidade que age por ou em lugar de outra ..." [6]

"uma entidade autônoma capaz de armazenar conhecimento sobre si próprio e sobre seu ambiente, que é capaz de definir objetivos e planos de forma a atingi-los." [3]

"processos orientados a tarefas que operam independentemente" [28].

"uma abordagem de programação que estende a noção de programação orientada a objetos, muitas vezes adicionando comportamento inteligente" [2]

Com base nestas definições conclui-se que agentes desempenham um papel ativo, originando ações que afetam o seu ambiente.

4.2 Atributos Fundamentais de Agentes

- **Persistência**: agentes mantêm um estado de consistência interno
- **Autonomia**: agentes exercem controle exclusivo sobre seu estado e comportamento interno
- **Reatividade**: agentes percebem e reagem a mudanças no seu ambiente
- **Habilidade de Comunicação**: agentes são capazes de trocar informações com outras entidades (agentes, humanos, objetos e seu ambiente)

4.3 Atributos Adicionais de Agentes

- **Iniciativa:** um agente exibe um comportamento oportunístico e não apenas reage a seu ambiente
- **Mobilidade**: agentes podem possuir habilidade de mover-se de um local para outro, preservando seu estado interno

- **Raciocínio**: um agente pode possuir a habilidade de inferir baseado no conhecimento e experiências - de uma forma racional e reproduzível

- **Habilidade de Planejar**: um agente pode sintetizar e escolher entre diferentes ações de forma a atingir seus objetivos.

- **Aprendizado e Adaptação:** um agente pode ser capaz de acumular conhecimento baseado em experiência passada e consequentemente modificar seu comportamento em resposta a novas situações

Com exceção de mobilidade, os atributos adicionais enumerados acima são usados principalmente em sistemas de agentes empregados para resolver problemas de inteligência artificial. Agentes para inteligência artificial são aqueles que exibem aspectos de comportamento inteligente.

4.4 Caracterização de Agentes

A aplicabilidade de agentes de software permeia diversas áreas da ciência da computação. Dependendo do domínio no qual está sendo utilizado, agentes podem possuir características diferentes. Uma aplicação clássica de agentes é em sistemas da Inteligência Artificial, onde os agentes são entidades que apresentam comportamento inteligente. Em outras áreas o comportamento inteligente não é uma característica essencial, e então o agente possui outras características adequadas à seu propósito. Por exemplo, em aplicações de Sistemas Distribuídos, o agente deve possuir mecanismos de comunicação para que ele possa comunicar-se com outros agentes em ambientes distintos. Apesar das diferentes aplicabilidades de agentes de software, eles compartilham algumas características comuns. Assim, por questões de generalidade e para oferecer uma visão global da tecnologia de agentes, serão analisadas nesta seção as características gerais de agentes, incluindo também àquelas não pertinentes ao foco deste trabalho. Porém, na análise de sistemas de agentes, seção 4.8, serão consideradas apenas as características importantes para comunicação dos agentes em um ambiente distribuído.

Agentes podem ser classificados segundo alguns critérios que caracterizam diferentes aspectos de sua implementação [ ]. Estes critérios são: método de ativação, localização de comportamento, tipo de conhecimento, origem do conhecimento.

4.4.1 Método de Ativação

Esta classificação refere-se à maneira como os agentes são disparados. Eles podem ser: chamados pelo usuário, chamados pela aplicação, disparados por eventos.

Agentes Chamados pelo Usuário são aqueles cujos serviços são explicitamente solicitados pelo usuário para realizar uma tarefa específica. Neste caso, geralmente o usuário pode visualizar e interagir diretamente com o agente.

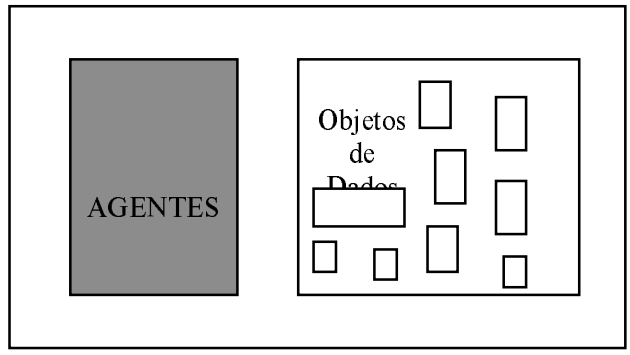
Agentes Chamados pela Aplicação são aqueles explicitamente invocados pela aplicação, geralmente sem a interferência do usuário.

Agentes Disparados por Eventos são aqueles ativados por eventos específicos que ocorrem durante a operação do sistema.

4.4.2 Localização de Comportamento

As figuras abaixo ilustram três tipos possíveis de agentes baseados na localização do comportamento do agente.

Sistemas Monolíticos com Agentes: os agentes são definidos independentemente dos objetos de dados mas todos agrupados para realizar determinadas tarefas, como mostrado na Figura 6.



Agentes Embutidos: os objetos incluem o conhecimento e a informação comportamental do agente, como mostrado na figura 7.

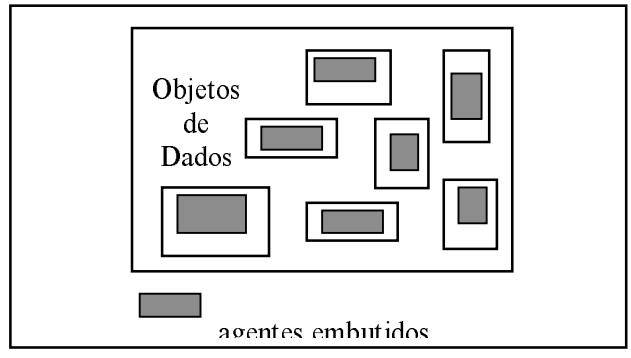


Figura 7 - Agentes Embutidos

Agentes independentes: podem ser vistos como uma coleção de processos com capacidades específicas que agem sobre uma determinada classe de objetos. Estes processos são definidos independentemente das instâncias dos dados e com tarefas independentes entre si, como mostrado na figura 8.

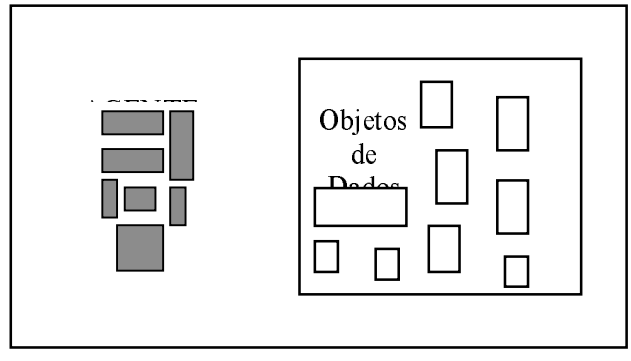


Figura 8 - Agentes Independentes

4.4.3 Tipo de Conhecimento

A complexidade do conhecimento controlada pelos agentes pode variar amplamente dependendo das técnicas utilizadas no seu desenvolvimento. Pode-se definir duas categorias:

Agentes Léxicos são entidades de software que executam ações dependendo do conteúdo dos objetos sendo tratados, mas não realizam interpretação do significado ou do relacionamento entre os dados. Um exemplo típico de agentes desta classe são aqueles que usam casamento padrão para sugerir caminhos em Browsers.

Agentes Semânticos utilizam uma descrição abstrata da estrutura da informação, geralmente em um domínio restrito, com a finalidade de aumentar a utilidade do agente podendo realizar inferências. Esta classe de agente é bastante utilizada nas aplicações de inteligência artificial.

4.4.4 Origem do Conhecimento

Pode-se classificar agentes de acordo com a origem do conhecimento utilizado por ele. Três tipos são possíveis:

Agentes cujo conhecimento é fornecido pelo programador: agentes que necessitam a intervenção do programador para atualizar sua base de conhecimento são menos flexíveis que os agentes das outras categorias. Estes agentes são úteis apenas para tarefas específicas como, por exemplo, para implementar atividades que não mudam durante longos períodos de tempo.

Agentes cujo conhecimento é fornecido pelo usuário: este tipo de agentes geralmente permite ao usuário definir e modificar a informação, geralmente em forma de regras, que determina o comportamento do agente. Esta especificação do comportamento é geralmente realizada usando *templates* ou uma linguagem orientada a agentes oferecida pelos próprios agentes ou pelo ambiente onde o agente opera.

Agentes que são capazes de aprender e dinamicamente adaptar-se às mudanças: esta é a forma mais adaptativa de agentes, que usa técnicas de aprendizado. Duas abordagens são sugeridas: aprender pela observação e aprender pela explanação. Na primeira abordagem, o agente infere padrões de comportamento diretamente observando uma série de ações do usuário e então é capaz de fazer um prognóstico e sugerir as ações futuras. Este tipo de aprendizado pode precisar de reforço do usuário para aumentar sua aquisição de conhecimento [20]. O aprendizado baseado na explanação é um método analítico que tenta formular generalizações de um único exemplo usando uma base de conhecimento previamente construída [ ].

4.5 O Programa Agente

O programa agente pode ser construído de componentes procedimentais ou de classes de objetos. Em qualquer um destes casos, o agente tem ligações a funções do seu ambiente de execução. Estas funções incluem funções importadas do sistema operacional, da aplicação ou de outros subsistemas, como por exemplo, outros programas agentes.

O programa pode ser executado em qualquer linguagem de máquina ou em uma linguagem interpretada (máquina virtual). Por questões de heterogeneidade, é preferível expressar o agente em uma linguagem interpretada. Isto penaliza a performance, mas como a maior parte do processamento de agentes não é feito no próprio agente, mas nas funções as quais ele é ligado, isto pode ser aceitável. Linguagens interpretadas também têm a vantagem de "late binding", o que possibilita que o agente contenha referências a funções ou classes não presentes no sistema onde ele se encontra, mas que estão disponíveis no sistema destino. Adicionalmente, linguagens interpretadas em geral oferecem maior segurança pois há o controle explícito de quais recursos do sistema são acessados pelos programas interpretados.

4.6 Comparação com outras abordagens

4.6.1. Programação Orientada a Agentes X Programação Orientada a Objetos

Programação Orientada a objetos é um mapeamento do problema no mundo real para o modelo computacional. Um objeto é um pacote de software auto-contido consistindo de sua própria informação privada (dados), seus próprios procedimentos privados (métodos privados) que manipulam os dados privados do objeto e a interface pública (métodos públicos) para comunicação com outros objetos [2]. Objetos são caracterizados por um *estado* e um *comportamento*. Eles interagem através de *mensagens*. Os princípios básicos da orientação a objetos são: encapsulação, herança e polimorfismo. A programação orientada a objetos surgiu com a finalidade de permitir interoperabilidade entre diferentes programas escritos por diferentes programadores. As linguagens orientadas o objetos oferecem esta interoperabilidade possibilitando o reuso de código via herança e encapsulação. Porém, os objetos vivem dentro de um único programa e dependem do compilador da linguagem que os criou.

A necessidade hoje vai além da interoperabilidade de objetos entre programas escritos na mesma linguagem de programação, utilizando o mesmo espaço de endereçamento. É necessário permitir interoperabilidade entre objetos criados por diferentes linguagens de programação, que executam em espaços de endereçamento, redes e sistemas operacionais diferentes. A solução que tem sido seguida é a extensão dos objetos clássicos com uma infra-estrutura adicional incorporando aos objetos características para viabilizar uma maior interoperabilidade. Na seção 4.8 serão analisadas as soluções apresentadas por alguns trabalhos neste sentido.

Programação orientada a agentes trata com primitivas de objetos e de agentes. Agentes podem ser pensados como meta-objetos pois eles também são caracterizados por um estado e comportamento. Diferentemente de objetos, agentes têm objetivos e trabalham para atingi-los.

Métodos de objetos são simplesmente procedimentos codificados pelo programador para tratar com diferentes mensagens. Por exemplo: um objeto cão pode latir quando provocado. Um agente, por outro lado, pode sintetizar planos para tratar com fatos não previstos pelo programador e pode aprender com a experiência. Por exemplo: um agente cão pode descobrir que latir insistentemente pode levá-lo a conseguir o alimento mais facilmente.

4.6.2. Agentes X Script

Um Script é um conjunto de ações pré-programadas usualmente executadas de maneira linear e determinística. Tipicamente tendem a ser escritos em linguagens procedimentais - um procedimento é geralmente uma diretiva que assume algumas pré-condições, realiza uma ação, e retorna um resultado. Linguagens procedimentais são adequadas para muitas tarefas, especialmente aquelas que têm um conjunto de soluções bem definidas. Mas não são bem adequadas para problemas que envolvem interação distribuída como ocorre entre múltiplos agentes em negociação. Em tais situações, a representação de situações, planos e objetivos são melhor representadas usando uma linguagem declarativa.

Agentes, em geral, são não determinísticos. Dado um conjunto com as mesmas condições e os mesmos objetivos, um script irá sempre produzir o mesmo resultado. Um agente, em particular quando é usado em aplicações de inteligência artificial, pode produzir diferentes resultados baseados no seu estado, crenças, etc.

Tipicamente, sistemas de agentes utilizados na computação distribuída são escritos em uma linguagem de script, como será discutido na seção 4.7.

4.6.3. Computação Baseada em Agentes X Modelo Cliente-Servidor

O modelo cliente-servidor caracteriza-se por dividir os sistemas em consumidores de serviços (clientes) e fornecedores (servidores). Existe um canal de comunicação usado para interação entre as duas partes.

A Computação baseada em agentes usa a metáfora do cliente-servidor mas também introduz um grau de paralelismo e assincronicidade.

Por exemplo, um banco de dados distribuído trabalhado no modelo cliente-servidor permite clientes remotos submeterem suas consultas ao servidor de banco de dados para processamento. Ele então retorna esses resultados pela rede para o cliente e o cliente volta a formular novas consultas. Este processo exige uma banda passante intensiva.

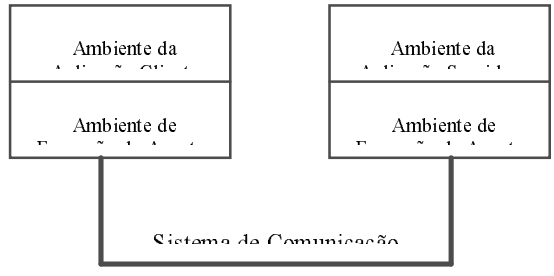
Um método mais eficiente poderia ter clientes enviando agentes de recuperação ao servidor e realizando uma extração. Tais agentes poderiam trazer a inteligência do cliente com ele e mover o processamento mais para perto da informação, de forma a diminuir o tempo e a banda passante.

Na próxima seção será discutida com mais detalhes a relação entre estes dois aspectos.

4.7 Agentes Móveis

São programas tipicamente escritos em uma linguagem de script que podem ser transferidos de um cliente para execução em um servidor remoto [6].

A figura 9 ilustra o conceito de agentes móveis.



Um computador cliente consiste de:

- um ambiente de aplicação, contendo uma ou mais aplicações que interagem com um servidor remoto. Através das APIs, a aplicação passa parâmetros aos programas agentes e vice-versa.

- um ambiente de execução de agentes, que tem a função de ligar os programas agentes com funções do sistema operacional e com o serviço de transporte de mensagens, para que agentes móveis possam ser enviados e recebidos via um sistema de comunicação.

- um subsistema de comunicação, que oferece as funções do serviço de transporte.

Para ilustrar o funcionamento de um programa com agentes móveis será descrita a migração da execução de um agente do cliente para o servidor:

. Um processo agente corrente é suspenso no ambiente de execução do agente ou um novo processo agente filho é criado.

2. O processo suspenso ou o novo processo, incluindo seu estado, pilha, "heap", e todas as referências externas são coletadas e incluídas em uma mensagem que deve ser expressa em uma linguagem de forma independente da máquina, por exemplo ASN. . [29]. Este passo é facilitado se o agente é um objeto de uma linguagem interpretada. Em particular, caso se saiba que classes idênticas existem também no destino, o agente pode ser reduzido a referências a objetos, instâncias de dados, e dados de estados do processo.

. A mensagem é endereçada ao destino final

4. A mensagem é roteada para o servidor destino onde ela é enviada para o ambiente de execução de agentes.

5. No ambiente de execução de agentes a mensagem recebida é reconstituída em um agente executável e o processo é acionado.

6. A execução continua na próxima instrução do programa agente.

A utilização de agentes móveis como base dos serviços e aplicações cliente-servidor adiciona capacidades às aplicações. Várias vantagens do uso de agentes no modelo cliente-servidor podem ser enumeradas. Para descrever uma típica situação real, nos exemplos abaixo [2] [6] [25] serão discutidas as vantagens do uso de agentes móveis sobre o uso de RPC. Isto porque a maioria das aplicações que podem ser realizadas por agentes móveis podem também ser realizadas via RPC.

A primeira é que na interação cliente-servidor utilizando agentes móveis, os agentes comunicam tanto dados como seus próprios procedimentos e exploram procedimentos residentes tanto no cliente quanto no servidor.

Outra vantagem é a redução do tráfego na rede. Como os agentes têm a habilidade de realizar suas ações no servidor e retornar ao cliente apenas a informação relevante, o tráfego na rede é minimizado. Comparando com a comunicação RPC, nesta há tipicamente vários fluxos entre o cliente e o servidor para realização de uma transação. Com o uso de agentes estes fluxos seriam incorporados em um único agente móvel que é transferido para o servidor, o que significa a redução do tráfego na rede.

A terceira vantagem é que o uso de linguagens de script fornece um melhor suporte para ambientes heterogêneos. O uso de uma linguagem de script para programa e troca de dados torna o programa e a representação de dados independente da plataforma, uma vez que o ambiente do script tenha sido instalado em todas as plataformas necessárias, como é o caso da linguagem Java que será comentada na seção 4.8.5.

O fato de transações baseadas em agentes evitarem a necessidade de preservar o estado do processo constitui a quarta vantagem do uso de agentes em aplicações cliente-servidor. A necessidade de preservar o estado do processo no cliente e no servidor durante cada fluxo de uma operação complexa adiciona considerável carga à computação cliente-servidor. A não ser que as aplicações cliente-servidor ofereçam métodos para tornar o estado persistente, o que adiciona um significante "overhead" na performance, a falha ou no cliente ou no servidor resulta na perda de no mínimo uma operação. Mecanismos de recuperação podem gerar muito tráfego na rede. A habilidade de agentes incorporarem seu estado libera a necessidade do cliente e do servidor preservarem estado do processo.

Apesar da mobilidade dos agentes ser uma capacidade especialmente importante para serviços e aplicações em redes, ela introduz sérios problemas de Segurança. Os Agentes móveis consistem em uma forma através da qual uma virose pode ser propagada na rede. A análise do agente para determinar se ele pode exibir um comportamento de um virus é um problema difícil, pois é difícil definir testes necessários e suficientes pelos quais o agente deve passar de forma a determinar se suas intenções são benignas ou se ele pretende infectar o sistema.

4.8. Análise de Trabalhos em Agentes de Software

Nesta seção serão discutidas algumas propostas de sistemas que utilizam agentes de software. A análise dos sistemas será conduzida mostrando como eles se comportam em relação aos seguinte parâmetros, que foram escolhidos devido a sua importância para interoperabilidade dos agentes do sistema.

- Qual o protocolo utilizado para a comunicação entre os agentes?

- Qual a linguagem usada para programação de agentes?

- O Controle da Distribuição é feito de que forma, ou seja, como é feita a localização dos objetos distribuídos?

- Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

- Qual a arquitetura seguida pelo sistema?

Eventualmente, serão comentadas características além das citadas acima, devido à importância especial que elas possuem para o sistema.

Uma crítica ao trabalho também será emitida onde serão citados aspectos não abordados pelo trabalho.

4.8.1. Um Esquema baseado em Agentes para permitir interoperabilidade [15]

Diferentemente dos demais trabalhos analisados, este trabalho aborda a questão da interoperabilidade de forma mais geral, oferecendo um esquema para desenvolvimento de software utilizando agentes. Este esquema propõe que os programas sejam escritos como agentes de software usando uma linguagem qualquer e a interoperação com outros programas seja feita através de uma linguagem de comunicação (ACL) e assistida por programas mediadores da comunicação (Facilitadores).

Segundo os autores, o trabalho fundamenta-se em idéias antigas na área de Sistemas Distribuídos que são:

- **o uso de padrões de comunicação**, que são frequentemente usados para interoperação de software dentro de áreas específicas, como, por exemplo, diferentes programas de correio eletrônico interoperam através do uso de padrões de correio como o SMTP (Simple Mail Transfer Protocol).

- **o uso de um mediador na comunicação**, que é uma abordagem já adotada por padrões de componentes de software, como, por exemplo, o ORB do CORBA é um mediador da comunicação entre componentes.

4.8.1.1 Qual protocolo é utilizado para a comunicação entre os agentes?

Os agentes não se comunicam diretamente uns com os outros, mas sim através de um *Facilitador*, que é um programa do Sistema Operacional responsável por assistir agentes em colaboração. Para um agente comunicar-se com outros, ele se comunica com seu facilitador local e este se comunica com os outros facilitadores.

Uma linguagem foi escrita especificamente para ser utilizada nesta comunicação. Tal linguagem é chamada ACL (Agent Communication Language). Na descrição do trabalho não é explicitado qual protocolo esta linguagem utiliza para dar suporte à comunicação em um ambiente distribuído.

4.8.1.2 Qual a linguagem é usada para programação de agentes?

Esta questão ficou em aberto, pois este trabalho é apenas uma proposta conceitual de um esquema para permitir a interoperação entre agentes em sistemas distribuídos, e não um Sistema em operação. No trabalho, são citados brevemente dois exemplos de implementações de ferramentas que tentaram utilizar a idéia proposta por Genesereth para concretizar a interoperação entre ferramentas de engenharia assistida por computador. A linguagem utilizada para implementar os módulos agentes não foi referenciada em nenhum dos dois exemplos.

4.8.1.3 O Controle da Distribuição é feito de que forma?

O controle da distribuição é realizado pelos Facilitadores. As mensagens enviadas pelos agentes não contém endereços. É de responsabilidade dos facilitadores rotear tais mensagens para o agente que possa

tratá-la de forma mais adequada. Isto é possível porque na fase de inicialização, o agente declara sua existência e comunica aspectos de sua especificação para o facilitador. Uma especificação consiste de sentenças na linguagem KIF (Knowledge Interchange Format) que descreve os interesses do agente e caracteriza seu comportamento. Baseado nestas sentenças o facilitador consegue inferir qual o agente mais adequado para tratar determinada mensagem enviada por outro agente.

Há uma estreita identificação das funções dos facilitadores com as do Object Request Broker (ORB) - o barramento de objetos do OMG, que será discutido na seção .4. Ambos são gerenciadores da comunicação entre as entidades comunicantes. Além da função de gerenciador da comunicação, o facilitador também funciona como repositório das especificações dos agentes em operação. É a mesma idéia do ORB que possui um Repositório de Interfaces dos objetos. O repositório do facilitador contém sentenças na linguagem KIF, enquanto o do ORB contém interfaces descritas em IDL (Interface Definition Language).

4.8.1.4 Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

A consistência das mensagens trocadas pelos agentes é garantida através do uso de uma linguagem comum para comunicação, que é a linguagem ACL. A linguagem ACL é uma linguagem declarativa baseada na idéia de que uma comunicação pode ser modelada através de sentenças declarativas. A mensagem em ACL tem a seguinte forma:

(Tipo da Mensagem, Conteúdo, Identificador da Mensagem).

De forma a garantir a mesma interpretação em agentes diferentes, são estabelecidos os possíveis tipos de mensagens trocadas entre os agentes que, segundo o autor, são suficientes para todas as comunicações entre agentes.

Os Tipos de Mensagens permitidos pelo sistema são:

Mensagens para comunicar ou retratar uma sentença declarativa:

(assert <p> <id>)

(retract <p> <id>)

Mensagens para fazer uma pergunta ou cancelar a pergunta:

(ask <p> <id>)

(unask <p> <id>)

Mensagens para sinalizar que a sentença transmitida no conteúdo deve ser setada verdadeira:

(achieve <p> <id>)

Mensagens para sinalizar que a sentença transmitida no conteúdo deve ser desprezada:

(forget <p> <id>)

Mensagens para responder a mensagens anteriores:

(reply <x> <id>) neste caso o x é o identificador da mensagem que fez o questionamento

Os argumentos de uma mensagem em ACL são expressões em uma linguagem formal chamada KIF (Knowledge Interchange Format).

Estes são os termos que, dentro da linguagem de comunicação, possibilitam que os demais agentes possam interpretar as ações da mesma maneira.

Além deste aspecto a nível da linguagem, pode-se identificar um outro aspecto que também trata da conformidade das mensagens. O facilitador oferece um serviço de transformação de uma mensagem de uma forma para outra, baseada na informação que ele tem sobre o agente selecionado para tratar a mensagem. Assim, ele age em um nível semântico, pois tem que fazer interpretações para poder realizar a tradução de forma conveniente.

4.8.1.5 Qual a arquitetura adotada no sistema?

O Sistema adota uma arquitetura federada (Figura 0) que une agentes e facilitadores. A denominação de arquitetura federada provém do fato de que os agentes formam uma "federação" na qual eles entregam sua autonomia de comunicação aos facilitadores. Desta forma, facilitadores consistem no ponto chave desta arquitetura, permitindo a liberdade de comunicação entre os agentes. Para realizar este trabalho, o facilitador realiza as seguintes funções:

- Determina o destino apropriado da mensagem indireta e envia tal mensagem para o destino. Fazendo isto, o facilitador funciona como um corretor (*broker*) de serviços oferecidos pelos agentes em sua comunidade. Neste caso, é empregada a mesma idéia utilizada no Object Request Broker (ORB) da OMG. Na arquitetura federada em questão, o facilitador desempenha funções semelhantes às que o ORB desempenha na arquitetura CORBA.

- Translação: a transformação de uma mensagem de uma forma para outra. Esta função é necessária devido às diferenças entre diferentes implementações de agentes. Por exemplo, um agente pode trabalhar com coordenadas cartesianas, enquanto outros com coordenadas polares. Na comunicação entre tais agentes o facilitador deve fazer a transformação de uma forma para outra. Para realizar esta função de translação os facilitadores usam mecanismos de raciocínio automático que ajudam a identificar a necessidade de realizar uma translação.

- Escalonamento: alocação de tempo para os jobs em cada processador em um sistema e também a alocação de jobs entre diferentes processadores.

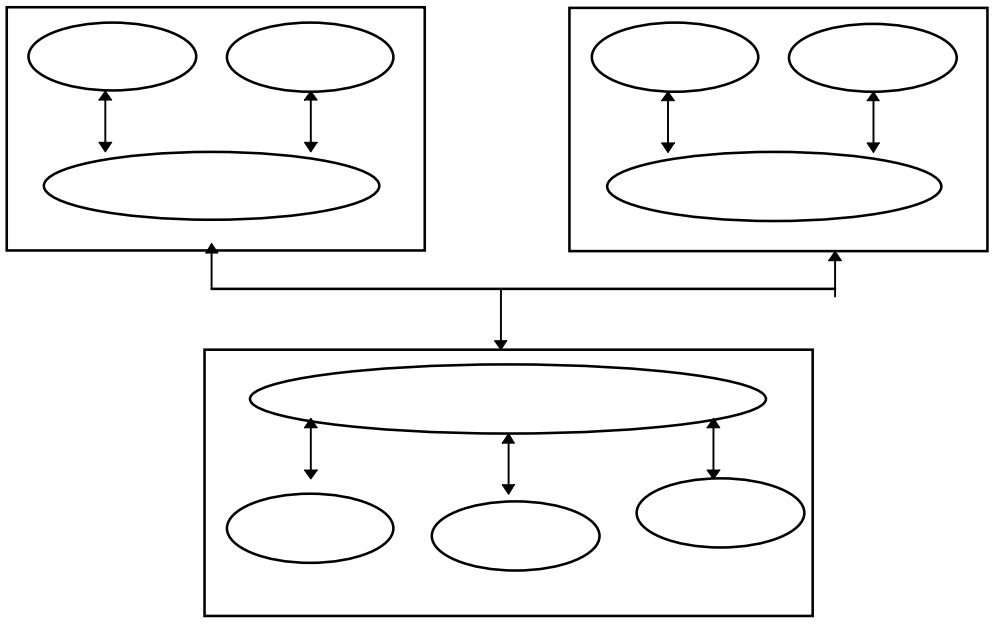


Fig.10 - AROUITETURA FEDERADA

4.8.1.6. Comentários

Neste trabalho, a solução para permitir interoperabilidade é fundamentada em dois aspectos:

- o uso do facilitador como intermediário na comunicação, adaptando as formas das mensagens e escolhendo o agente mais adequado para tratar a mensagem

- comunicação usando um padrão: a linguagem ACL com sentenças em KIF

O trabalho tenta integrar a idéia de agentes com a proposta de usar um barramento de software aberto adotada nos padrões para componentes de software. Tanto o facilitador quanto o ORB têm a responsabilidade de assegurar que as entidades de software interoperem corretamente, mas a proposta da OMG parece ser mais geral, pois neste caso os objetos componentes poderão ser escritos por diferentes vendedores e em linguagens diferentes. Além disto, o ORB permite chamar métodos em objetos usando a linguagem de alto nível de preferência, enquanto que neste esquema proposto por Genesereth, a única linguagem para comunicação é ACL. O facilitador oferece uma vantagem em relação ao ORB que é a capacidade de fazer inferências sobre uma mensagem enviada por um agente durante a procura de outro agente que possa satisfazê-la adequadamente. A inferência é possível porque as sentenças são escritas em KIF e porque os facilitadores usam mecanismos de raciocínio automático. Os autores alegam que a idéia de facilitador já existe em outras propostas para interoperabilidade, mas que a novidade neste trabalho é o uso de facilitadores com raciocínio automático, o que possibilita o casamento de clientes e servidores de forma mais abrangente.

Um aspecto interessante proposto neste trabalho é a possibilidade de eliminar o facilitador na comunicação entre dois agentes sempre que o facilitador reconhece que haverá uma frequente comunicação entre dois agentes. O facilitador identifica a estreita relação entre os agentes na fase de inicialização do agente. O exemplo citado no trabalho envolve dois agentes, um banco de dados e um editor de banco de dados. Supondo que estes dois agentes são os únicos interessados nas mensagens um do outro, o facilitador pode solicitar aos agentes que implementem sua comunicação usando Chamada

Remota de Procedimento (RPC), fornecendo o endereço do outro agente e deixando de ser o intermediário na comunicação. Esta facilidade visa eliminar o "overhead" introduzido pelo facilitador que, antes de passar a mensagem de um agente para outro, realiza suas funções de roteamento, translação e escalonamento.

Para evitar inconsistências durante a comunicação direta entre os agentes, o facilitador pode solicitar aos agentes que terminem ou modifiquem as conexões, pois alguma mudança ocorreu no ambiente (mudança na configuração do sistema, mudança no interesse de algum agente, introdução de novo agente com interesse relacionado) determinando mudanças também nas conexões.

Apesar de ter como propósito apresentar uma arquitetura geral a ser seguida por diferentes sistemas que utilizem a idéia de agentes para interoperação entre programas, o trabalho não esclarece algumas questões importantes para a compreensão do funcionamento de uma possível implementação.

Questões não esclarecidas:

- Os Agentes em comunicação têm de ser escritos na mesma linguagem? De acordo com a idéia de uma linguagem de comunicação comum para os agentes em interação, subtende-se que os agentes podem ser escritos em linguagens diferentes e na comunicação eles usam esta linguagem para que a mensagem seja interpretada da mesma forma pelos parceiros de comunicação.

- O facilitador está a nível de sistema operacional e interage com agentes usando ACL.

    - O facilitador deve ser implementado na mesma linguagem dos Agentes? Supõe-se que não deve haver esta restrição pois como os agentes se comunicam com o facilitador usando uma linguagem padrão, eles podem ser escritos em linguagem diferentes.

    - Qual o protocolo usado para comunicação entre os facilitadores?

- ACL é uma linguagem de comunicação. Qual o protocolo usado pela linguagem para envio de mensagens através da rede?

4.8.2. KQML: Knowledge Query and Manipulation Language [9] [24]

Este trabalho aborda a questão da interoperabilidade entre agentes inteligentes em uma aplicação distribuída. O KQML é um produto do KSE (ARPA Knowledge Sharing Effort) que é um consórcio para desenvolvimento de convenções para facilitar o compartilhamento e reuso de bases de conhecimentos e de sistemas baseados em conhecimento.

O KQML consiste em uma linguagem e um protocolo para permitir a interoperabilidde entre agentes inteligentes em uma aplicação distribuída. A linguagem apresenta um formato para as mensagens que serão trocadas e o protocolo efetiva o compartilhamento entre agentes através desta troca de mensagens.

O interesse do KQML é permitir usuários de aplicações interfacearem com sistemas baseados em conhecimento e efetivarem uma comunicação com troca de conhecimento.

O KQML divide a linguagem que é utilizada para trocar conhecimento em duas partes:

- a linguagem usada para expressar o conteúdo da informação a ser trocada. O KQML permite que se use qualquer linguagem para expressar o conteúdo.

- a linguagem de comunicação específica usada para trocar as expressões de conteúdo. É a linguagem usada para encapsular as mensagens, roteá-las, e fazer a ligação entre emissores e receptores. É a este aspecto que o KQML oferece suporte.

4.8.2.1 Qual protocolo é utilizado para a comunicação entre os agentes?

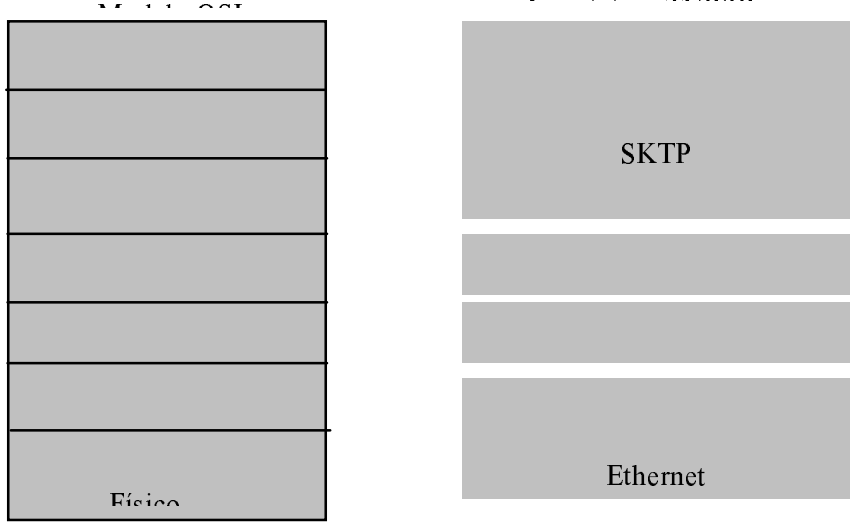
A comunicação entre os agentes é o ponto chave tratado pelo KQML. Conceitualmente, pode-se alegar que a comunicação no KQML é um processo constituído de etapas:

- um processo é responsável por empacotar a mensagem de forma adequada
- outro processo tem como objetivo localizar e engajar o agente com o qual a comunicação vai ser estabelecida.

Para empacotamento da mensagem a ser transmitida, o KQML oferece um padrão para formato de mensagem que encapsula tanto o conteúdo a ser transmitido quanto informações sobre este conteúdo.

Para localizar o agente destino e efetivar esta comunicação, o KQML oferece um protocolo a nível de aplicação da arquitetura Internet TCP/IP, chamado SKTP (Simple Knowledge Transfer Protocol), conforme mostra a Figura 11.

O objetivo principal do SKTP é esconder as primitivas de comunicação do programador da aplicação.



Do ponto de vista da aplicação ela estará se comunicando com outra aplicação usando o SKTP, sem tomar conhecimento do protocolo que dá suporte a sua comunicação (TCP/IP).

O SKTP efetiva interações KQML, ou seja, transmite mensagens no formato padrão KQML e é subdividido em três camadas (Figura 1).

As expressões KQML consistem de uma expressão de conteúdo encapsulada em uma mensagem que por sua vez é encapsulada em uma estrutura de comunicação. Assim, a linguagem KQML define uma estrutura para representar as expressões de cada camada:

- **Conteúdo:** contém uma expressão em qualquer linguagem. O KQML não trata deste aspecto.
- **Mensagem:** usa as expressões KQML para codificação da mensagem que a aplicação transmite a outra. Como o conteúdo é indiferente ao KQML, esta camada adiciona um conjunto de características que descreve o conteúdo, como: a linguagem na qual o conteúdo é expressado, a ontologia que ela

assume (a categorização do conteúdo, ou seja, sobre que assunto o conteúdo trata), o tipo de ato de fala que o conteúdo representa (declaração, pergunta, etc.).

- **Comunicação:** adiciona características à camada de mensagens que especifica os parâmetros de comunicação como identidade do emissor e receptor, identificador único para a mensagem, etc. Neste nível agentes trocam pacotes.

Figura 12 - Pilha de Protocolos SKTP

4.8.2.2 Qual a linguagem é usada para programação de agentes?

Quando usando o KQML, um agente transmite o conteúdo composto em qualquer linguagem de sua escolha, empacotado em uma mensagem KQML. O KQML ignora o conteúdo da mensagem. Ou seja, existe uma distinção entre a linguagem de conteúdo, que expressa fatos sobre determinado domínio, e a linguagem de comunicação (KQML) , que expressa atos comunicativos. A vantagem desta separação é que facilita a integração do KQML em aplicações que utilizem as mais diversas linguagens para expressar seu conteúdo. Ou seja, o agente pode ser programado em qualquer linguagem, mas quando engajado na comunicação, ele deve usar o KQML para empacotar sua mensagem e transmiti-la usando o SKTP.

A sintaxe do KQML é baseada no Lisp e é definida por uma lista entre parêntesis, com um conjunto de argumentos obrigatórios associados (que inclui o conteúdo real da mensagem) e com um conjunto de argumentos opcionais. O formato das expressões KQML são pares parâmetro-valor. O KQML define o argumento TYPE que declara qual o tipo da mensagem que está sendo transmitida no pacote. Para instanciação do argumento TYPE são definidos pelo KQML vários possíveis tipos de mensagens. Estes tipos de mensagens têm a função de determinar o que o agente destino deve fazer com as sentenças da linguagem de conteúdo.

O exemplo abaixo ilustra uma comunicação usando KQML. Um pacote é enviado da aplicação ap001 para a aplicação ap00 , no modo de comunicação com bloqueio, contendo uma mensagem do tipo *query* , cujo conteúdo expressado na linguagem KIF é *color snow _C*. O campo *Qualifiers* indica que para esta mensagem apenas uma resposta é esperada.

(PACKAGE

: FROM ap001

: TO ap00

: ID DVL-f001-111791.101 91

: COMM block

: CONTENT

(MSG

: TYPE query

: QUALIFIERS (:NUMBER-ANSWERS 1)

: CONTENT-LANGUAGE KIF

: CONTENT (color snow _C)))

Apesar de apresentar a flexibilidade do uso de agentes programados em diferentes linguagens, apresenta a restrição de permitir interação apenas entre agentes programados na mesma linguagem.

4.8.2.3 O Controle da Distribuição é feito de que forma, ou seja, como é feita a localização dos objetos distribuídos?

A localização dos objetos é feita pelo Facilitador. Cada aplicação está associada com uma biblioteca para interface com o facilitador (FIL) e com um facilitador. O FIL recebe da aplicação uma mensagem na linguagem da aplicação, forma a mensagem KQML e entrega ao facilitador. O facilitador aceita mensagens de FILs e analisa a informação nos campos da mensagem para determinar os destinos apropriados para ela. Em alguns casos uma aplicação pode identificar um site particular como sendo o destino da mensagem pelo nome do host (To: oxum.inf.puc-rio.br) ou mais simbolicamente (qualquer máquina no site inf.puc-rio.br que tenha um servidor de arquivos). Em outros casos, uma aplicação pode não saber qual é o site apropriado, então o facilitador deve recorrer aos valores de outros campos da mensagem e ao conhecimento de outros sites para decidir para onde rotear a mensagem.

Neste modelo de sistema, todas as aplicações oferecem a seus FILs as declarações de perguntas ou afirmações que elas podem importar ou exportar e então, seus facilitadores associados irão transmitir estas informações pela rede.

Para realizar sua tarefa cada facilitador mantém seu próprio Banco de Dados para as aplicações. Cada entrada no Banco de Dados oferece um endereço Internet do host onde a aplicação está executando e uma porta para o facilitador neste host. O Banco de Dados é mantido por facilitadores ativos usando as seguintes regras:

- Quando uma aplicação local declara ao seu facilitador que está pronta para importar perguntas ou afirmações, o facilitador dissemina esta informação para todos os sites que estão executando um facilitador.

- Quando um facilitador recebe uma declaração de outro facilitador, ele confirma o recebimento enviando uma lista de declarações que suas aplicações estão prontas para aceitar.

O processo realizado pelo facilitador não é difícil mas há vários problemas que exigem trabalho, especificamente se o número de agentes disponível na rede cresce e a complexidade da informação trocada também cresce. Diversas abordagens podem ser utilizadas de forma a organizar melhor o trabalho do facilitador e diminuir sua complexidade. Uma questão importante é: Onde e como o Banco de Dados deve ser implementado? Há várias alternativas:

1. Pode ser replicado em cada facilitado;

. Pode ser centralizado em uma máquina em cada site;

3. Pode ser armazenado de forma distribuída pela rede.

A estratégia de implementação utilizada é baseada nos requisitos do ambiente.

Facilitadores oferecem um esquema declarativo nos quais aplicações especificam o conhecimento necessário e os serviços que oferecem, estabelecem canais de comunicação entre agentes apropriados e mediam o diálogo resultante.

O seguinte exemplo ilustra como o facilitador trabalha:

TYPE: query

LANGUAGE: Prolog

ONTOLOGY: DRPI-93

TOPIC: Airports: Locations

Se o facilitador recebe uma mensagem com as informações acima, ele deve procurar por um ou mais sistemas que tenha declarado que deseja Importar perguntas deste tipo. Esta procura é feita no Banco de Dados de declarações, observando as entradas que casam com esta questão. Quando ele encontra as entradas, envia uma mensagem aos facilitadores que as representam e espera pela resposta.

Como no trabalho discutido em .8.1, este trabalho também usa a idéia de facilitador que, por sua vez, é semelhante ao ORB do OMG. Na sua função de repositório de especificações de agentes em operação, o facilitador proposto neste trabalho contém declarações na linguagem KQML, enquanto que o ORB contém interfaces descritas em IDL e o facilitador descrito em [15] contém sentenças na linguagem KIF.

4.8.2.4 Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

O uso das expressões KQML garante que os facilitadores interpretam a mensagem de forma correta e usa os argumentos contidos nela para procurar no seu banco de dados o agente mais adequado para tratá-la.

Os argumentos da mensagem KQML que são usados para permitir o casamento correto entre os agentes são:

LANGUAGE: a linguagem na qual o conteúdo é expressado

TYPE: “query”, “assertion”. “definition”, etc.

ONTOLOGY: o contexto no qual o emissor da mensagem está inserido e no qual o agente receptor também deve estar inserido

TOPIC: o assunto específico da mensagem. Este campo pode ser apenas interpretado no contexto de uma dada ontologia.

Com base nestas informações, apenas são efetivadas interações entre agentes que usam a mesma linguagem para expressar o conteúdo, que estão inseridos na mesma ontologia e tratando do mesmo assunto. Além disso, a interação só é efetivada quando o agente destino declara poder receber mensagem do tipo que está sendo enviada.

Desta forma, a própria estrutura oferecida pelo KQML e a política do mecanismo de inferência usado para casar os agentes garantem que os agentes comunicantes interpretam as mensagens da mesma forma.

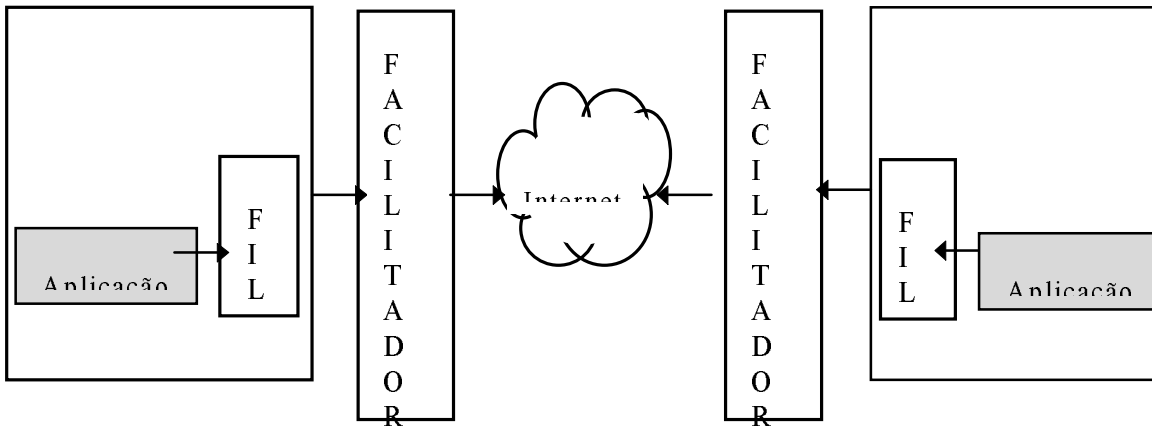


Figura 13 - Arquitetura de Implementação do SKTP

Esta estratégia é restritiva pois permite interação apenas entre agentes que usam a mesma linguagem para expressar o conteúdo. Assim, a interoperabilidade se dá apenas entre agentes que trabalham com a mesma linguagem.

A idéia de utilização de uma linguagem comum para comunicação foi proposta pelo trabalho de Genesereth [15] e absorvida por este trabalho. Naquele trabalho a linguagem de comunicação chamada ACL define um formato de mensagem para ser utilizada na comunicação entre agentes. Da mesma forma, a idéia de tipos de mensagens também é utilizada no trabalho de Genesereth. Naquele trabalho os argumentos da mensagem ACL são expressões na linguagem KIF, enquanto neste, os argumentos são expressões em KQML.

4.8.2.5 Qual a arquitetura seguida pelo sistema?

A arquitetura de implementação do SKTP está ilustrada na Figura 13.

O SKTP oferece uma interface entre a aplicação e o protocolo de comunicação. Cada componente da arquitetura tem o equivalente com a pilha de protocolos mostrada na Figura 1 .

A camada de mensagens é implementada pelo módulo FIL (Facilitator Interface Library), ou seja, é oferecida uma biblioteca de Interface com o Facilitador.

A camada de comunicação é implementada pelo Facilitador que é responsável pelo roteamento de mensagens. Várias aplicações podem compartilhar um mesmo facilitador. O protocolo TCP/IP é responsável pelo transporte de dados entre os facilitadores.

O FIL é o código que conecta as diferentes linguagens de Inteligência Artificial para comunicação com o mundo KQML. O FIL realiza 3 funções:

- intercepta um conjunto de declarações que descreve as transações internas da base de conhecimento que devem ser importadas ou exportadas para sistemas remotos
- contém código que monitora as transações internas e organiza as expressões que serão transmitidas como mensagens a um facilitador que irá roteá-las apropriadamente
- contém código que oferece acesso ao facilitador para envio de mensagens

O FIL é fortemente integrado com a aplicação, por isto ele é parcialmente implementado na linguagem de implementação da aplicação. Ele precisa ser parcialmente implementado na linguagem de implementação da aplicação porque ele acessa as rotinas internas da linguagem para ajudar a determinar

quando expressões precisam ser transmitidas e quais expressões devem ser selecionadas. Por exemplo, existe uma implementação do SKTP em Common Lisp e um protótipo que tem aplicações escritas em Prolog. O FIL para estas aplicações é escrito parcialmente em Prolog e parcialmente em Common Lisp.

Quando uma aplicação declara que precisa exportar mensagens para agentes remotos, o FIL cria código para monitorar a geração da mensagem KQML. O conteúdo é encapsulado como uma mensagem e passada ao facilitador. O FIL pode esperar por respostas ou não. Esta e outras decisões são tomadas pelo *designer* do FIL e podem ser passadas como opções a serem escolhidas pelo programador da aplicação.

Se uma aplicação está pronta para receber mensagens de agentes remotos, ela declara da mesma forma que declara para exportar expressões. Porém, neste caso, o FIL tem de estabelecer um conjunto de pontos de entrada para o qual um facilitador pode enviar as mensagens. A implementação desta conexão depende do design do facilitador e do tipo de conexão que ele tem com os FILs. Na implementação, o facilitador pode ser parte do FIL ou pode ser um processo separado conectado por memória compartilhada ou algum tipo de canal de comunicação entre processos.

Facilitadores são agentes que oferecem serviços como servidores de nomes, brokers, autenticadores, etc.

4.8.2.6 Comentários

Pode-se identificar uma estreita identificação do trabalho do KQML com o esquema proposto por Genereseth [15] discutido no item anterior. Em linhas gerais, pode-se dizer que o KQML é uma implementação das idéias sugeridas naquele trabalho, adicionando suas características particulares.

Igualmente ao trabalho de Genereseth, a solução para permitir interoperabilidade é baseada no uso de uma linguagem padrão para comunicação e de um facilitador que é o mediador desta comunicação.

O KQML é um protocolo de alto nível independente do mecanismo de transporte, ou seja, apesar de ter originalmente sido implementado para usar o TCP/IP, ele pode ser usado com outros protocolos como SMTP (email), HTTP, etc. A flexibilidade das expressões KQML conferida pelos argumentos opcionais permite a variação do formato da mensagem, o que pode facilitar o uso de diferentes mecanismos de transporte para transmissão de mensagens KQML. Por exemplo, em implementações que usam TCP streams, os parâmetros de comunicação como identificador do emissor e receptor são campos do corpo da mensagem KQML. Em uma implementação de Correio Eletrônico para transmissão de mensagens KQML, estes campos podem ser colocados no cabeçalho da mensagem de correio e não na mensagem KQML. Ou seja, a sintaxe e o conteúdo da mensagem KQML pode variar dependendo do mecanismo de transporte utilizado.

Segundo Mayfield [ ], como o KQML é baseado no uso de um facilitador, ambientes KQML podem ser facilmente adaptados para usar módulos que desempenham a mesma função do facilitador oferecidos em produtos comercialmente disponíveis como o CORBA ou o OLE.

Uma questão importante para sistemas que trabalham em um ambiente distribuído é a segurança. Como os mecanismos de transporte são inerentemente não confiáveis, fica a cargo dos programas de aplicação prover esta confiabilidade. As questões de segurança e autenticação estão começando a receber atenção especial na comunidade envolvida com pesquisas no KQML. Um modelo de uma arquitetura de segurança baseado em técnicas de criptografia é proposto em [35]. A arquitetura proposta oferece dois modelos de segurança, básico e avançado. O modelo básico de segurança oferece suporte para autenticação do emissor, integridade da mensagem e privacidade dos dados. O modelo avançado cobre a não repudiação da origem e proteção contra ataques na resposta da mensagem. Este modelo realiza mudanças freqüentes nas chaves usadas na criptografia para evitar ataques.

4.8.3. AgenTalk [22]

O AgenTalk é uma linguagem para facilitar o desenvolvimento de protocolos de coordenação entre agentes autônomos. Protocolo de Coordenação é o protocolo usado para agentes trocarem mensagens.

Neste sistema a definição de protocolos de coordenação pode ser feita incrementalmente através do mecanismo de herança.

4.8.3.1 Qual protocolo é utilizado para a comunicação entre os agentes?

Este é o ponto chave da proposta do AgenTalk. Como no KQML, agentes coordenam suas ações trocando mensagens e esta comunicação com troca de mensagens se dá através de um protocolo de coordenação de alto nível. Diferentemente do KQML, no AgenTalk cada sistema de agente pode utilizar um protocolo de coordenação distinto. O AgenTalk é uma linguagem de programação para implementação de protocolos e agentes que seguem o protocolo. A sua política é permitir a definição de novos protocolos baseados em protocolos existentes e também possibilitar a customização de protocolos existentes para que seja possível adaptá-lo de acordo com o domínio da aplicação.

Como são definidos protocolos de coordenação no AgenTalk? Um protocolo é definido por uma máquina de estado finito representada por um script. Usando o modelo de uma máquina de estado finito, são definidos estados do protocolo de coordenação e, para cada estado, são definidas regras de transição que podem determinar ações no script ou no agente. Como na programação orientada a objetos, é definido um mecanismo de herança para scripts através do qual protocolos podem ser definidos incrementalmente herdando a definição de um script existente.

A customização do protocolo é possível através da redefinição de funções no script. Para cada regra de transição no script é oferecida uma interface de programação que permite cada aplicação particular customizar o protocolo. Através desta interface, o script pode chamar uma função *callback* definida no agente que o chamou ou uma função padrão definida no script. Quando um agente vai seguir determinado protocolo, ele invoca o script correspondente ao protocolo. No script, quando uma regra de transição for disparada, pode ser que a ação correspondente seja chamar a função no agente definida na interface desta regra. Assim, o mesmo script pode causar diferentes comportamentos em agentes diferentes. No caso do agente não ter uma função definida para determinada regra de transição, uma função do script com o mesmo nome é chamada quando tal regra for disparada. Isto permite que exista sempre um comportamento padrão do script que pode ser adaptado quando for conveniente para a aplicação.

Os protocolos de coordenação utilizam o protocolo TCP/IP para o transporte das mensagens.

4.8.3.2 Qual a linguagem é usada para programação de agentes?

A linguagem utilizada para programação de agentes é a AgenTalk ou qualquer outra linguagem que possua um mapeamento para o AgenTalk, permitindo que agentes escritos em diferentes linguagens possam interagir com agentes escritos em AgenTalk. Na realidade o foco do AgenTalk é a implementação do protocolo de coordenação e não de detalhes de programação do agente. Mas a utilização do protocolo implica em interação entre o agente e o protocolo escrito em AgenTalk. Assim, para o agente programado em outra linguagem usar o protocolo de coordenação escrito em AgenTalk, esta linguagem utilizada no agente tem de ter um mapeamento para AgenTalk.

4.8.3.3 O Controle da Distribuição é feito de que forma, ou seja, como é feita a localização dos objetos distribuídos?

No AgenTalk, quando um agente é criado, é chamada uma função *add_agent* que o registra em um *post-office*. Um post-office é uma estrutura que armazena uma lista de agentes. Quando um agente deseja se comunicar com outro, ele tem de explicitamente endereçar o agente destino ou endereçar o post-office destino. O nome do agente é dado na forma nome-local@endereço-do-host.

Quando um agente deseja encontrar um outro agente para realizar um serviço desejado, ele distribui solicitações para todos os post-offices (broadcast). Os post-offices enviam a mensagem para os agentes registrados nele. Depois de avaliar as solicitações, os agentes capazes de atendê-las submetem suas ofertas ao agente de origem, que então vai decidir qual agente irá realizar o serviço.

A desvantagem desta abordagem é que irão existir muitas comunicações para cada solicitação desejada, pois não existe a nível de post-office uma busca para encontrar qual agente mais adequado para atender uma solicitação. Todos os agentes vão receber a mensagem e tentar fazer um casamento com as suas funções.

4.8.3.4 Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

O sistema só permite comunicação entre agentes escritos em AgenTalk e o protocolo é customizado de acordo com a aplicação. Como dois agentes em interação utilizam o mesmo protocolo, então eles utilizam as mensagens definidas neste protocolo, e portanto sabem interpretá-las convenientemente.

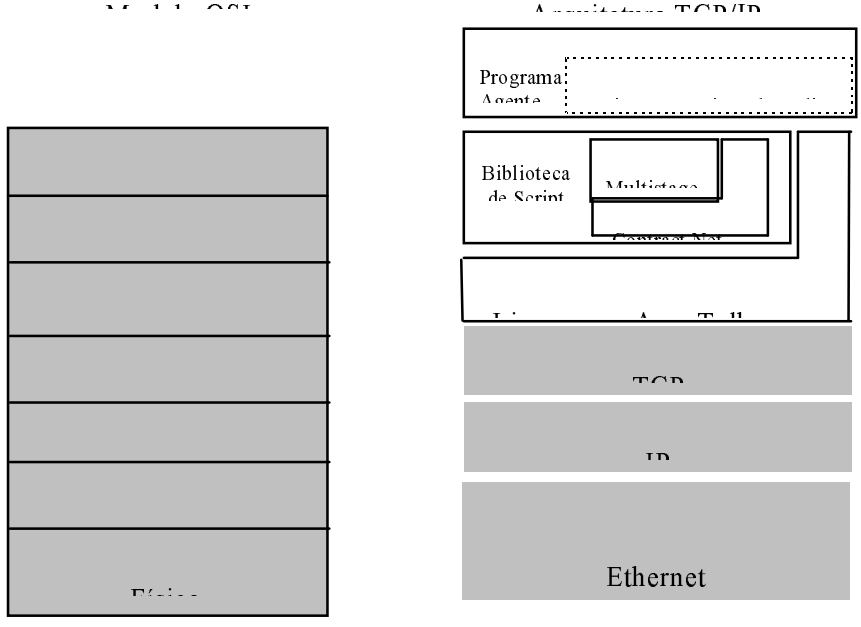
4.8.3.5 Qual a arquitetura seguida pelo sistema?

A arquitetura seguida pelo sistema está ilustrada na figura 1 .

Esta figura ilustra a disposição da linguagem AgenTalk e dos protocolos gerados a partir dela seguindo usando na arquitetura TCP/IP.

O programa agente usa um protocolo de coordenação definido por um script que pode ser baseado na biblioteca de scripts existente ou em algum protocolo pré-definido como o Contract Net Protocol ou o Multistage protocol. Opcionalmente, o programa pode definir um script totalmente novo, independente dos scripts já existentes.

Diferentemente dos sistemas de agentes discutidos anteriormente, nesta arquitetura não são ilustrados aspectos da comunicação entre agentes distintos. Sua ênfase é na criação do protocolo de coordenação.



4.8.3.6 Comentários

Neste sistema, a solução para permitir interoperabilidade é baseada no uso de um protocolo comum para comunicação entre os agentes e na possibilidade da customização de um protocolo para adequar-se ao domínio da aplicação.

O AgenTalk preocupa-se bastante em oferecer mecanismos para a definição destes protocolos, deixando em aberto muitas questões importantes para interoperabilidade. Por exemplo, no trabalho de Genesereth o agente comunica aspectos de sua especificação para o facilitador, por isto é possível que o facilitador realize inferências para encontrar um agente capaz de tratar determinada solicitação. Da mesma forma, no KQML existe a idéia do facilitador como intermediário na comunicação, e este facilitador contém um banco de dados associado que é consultado sempre que um agente envia uma solicitação ao facilitador. No AgenTalk, o post-office, que é uma estrutura que agrupa a identificação dos agentes existentes, não tem associado a ele nenhuma informação adicional sobre o agente que o permita realizar alguma consulta prévia de forma a evitar que todos os agentes sejam incomodados quando uma solicitação é enviada ao post-office. Assim, o post-office apenas direciona a mensagem para todos os agentes registrados nele.

As bibliografias disponíveis sobre este sistema não são muito esclarecedoras e deixam muitas questões em aberto, como:

- Quando o post-office é criado?
- O post-office é uma estrutura definida para cada aplicação particular, para todas as aplicações que estão executando em uma determinada máquina ou para um ambiente composto por um determinado número de máquinas?
- Como é feita a localização dos post-office?

4.8.4. Obliq [7]

A linguagem Obliq é uma linguagem interpretada baseada em objetos sem o conceito de hierarquia, que apresenta um modelo para programação com agentes distribuídos, onde objetos podem trabalhar remotamente e agir como representantes do seu site original.

Um agente escrito em Obliq pode ser totalmente autônomo, ou seja, ter liberdade de agir em qualquer lugar da rede onde exista um interpretador Obliq, ou conter *identificadores livres*, introduzidos pelo Obliq para controlar as ações do agente.

O conceito de *escopo léxico distribuído,* em conjunto com o conceito de identificadores livres, consistem nos pontos chave para o suporte a computações distribuídas através de agentes em Obliq. Um agente com identificador livre tem um escopo léxico associado, que significa que este agente pode ser livremente transmitido pela rede mas retém suas ligações com seu site original. Ou seja, os identificadores livres transmitidos no agente são limitados à sua localização original mesmo se esta localização pertence a um site diferente daquele onde o objeto se encontra. Através destes identificadores, agentes migrantes mantém conexões com objetos em vários sites. Assim, para ser completamente auto-contido, o agente não deve ter identificadores livres, pois, desta forma a execução de tal agente pode ser realizada autonomamente no servidor remoto. Esta flexibilidade torna possível a alocação de objetos em sites remotos que podem agir como Agentes do site origem.

Obliq é uma linguagem não tipada, ou seja, não há tipagem estática. Segundo seus projetistas, esta decisão foi tomada em face do seu propósito de ser uma linguagem para programação distribuída, pois desta forma os processadores são mais simples e menores e podem ser facilmente embutidos em aplicações. Adicionalmente, programas não tipados são mais fáceis de distribuir evitando problemas de compatibilidade de tipos em diferentes sites. Em contraste, o Obliq *runtime* é fortemente tipado.

4.8.4.1 Qual protocolo é utilizado para a comunicação entre os agentes?

O TCP/IP é protocolo de transporte usado para transmissão de agentes pela rede. Não há um protocolo de comunicação de alto nível como existe no KQML e no AgenTalk.

4.8.4.2 Qual a linguagem é usada para programação de agentes?

A linguagem usada para programação de agentes é a Obliq. Aspectos de interoperabilidade com agentes escritos em outra linguagem não são tratados em Obliq.

4.8.4.3 O Controle da Distribuição é feito de que forma, ou seja, como é feita a localização dos objetos distribuídos?

A localização dos objetos distribuídos é feita via um servidor de nomes. Em Obliq, o servidor de nomes é um processo externo, identificado por um endereço IP que mantém uma tabela associando nomes de objetos com referências de rede.

A linguagem Obliq oferece um módulo, chamado *net,* que contém funções para controle da distribuição. Um objeto que pode ser exportado é registrado em um servidor de nomes através da função *net_export*. Quando um cliente deseja importar um objeto, através da função *net_import* ele consulta o servidor de nomes e obtém uma referência de rede para o objeto cujo nome foi passado como parâmetro.

Desta forma, a linguagem Obliq oferece pouca flexibilidade em relação à localização de objetos distribuídos. O programador tem a responsabilidade de identificar qual agente lhe pode ser útil e também de explicitamente se comunicar com este agente. O servidor de nomes é um intermediário passivo na comunicação, com função bem diferente dos Facilitadores existentes na proposta de Genesereth e do KQML e do ORB do CORBA. Como o nome sugere, ele é apenas um repositório de nomes de objetos com sua respectiva localização de rede, sem nenhuma informação adicional que permita um corretor (*broker)* agir para encontrar um agente adequado para realizar determinada tarefa. Pode-se identificar uma similaridade entre as funções deste servidor de nomes e do *port-mapper* do RPC [10]. O *port-mapper* usa um servidor para manter um banco de dados que mapeia número de programas RPC a portas. Ou seja, da mesma forma que o servidor de nomes do Obliq, o port-mapper do RPC consiste em um repositório de nomes com informações de localização associadas, só que no primeiro caso são nomes de objetos e sua localização de rede e no caso do port-mapper são nomes de programas RPC com sua respectiva porta.

4.8.4.4 Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

Como o sistema só endereça migração e comunicação de agentes escritos em Obliq executando em plataformas contendo o interpretador Obliq, a semântica das operações definidas no agente deve ser interpretada corretamente pelos outros agentes.

4.8.4.5 Qual a arquitetura seguida pelo sistema?

Pode-se situar o presente trabalho no nível de aplicação da arquitetura TCP/IP.

4.8.4.6 Comentários

A interoperabilidade neste sistema é restrita a agentes em plataformas arbitrárias que usem a mesma linguagem de programação, a linguagem Obliq. Como o Obliq é uma linguagem interpretada, seus programas são independentes de plataforma e podem executar em ambientes onde exista o interpretador Obliq. Assim, este sistema não trata aspectos de interoperabilidade no sentido amplo, mas dá suporte ao acesso e envio de objetos Obliq através da rede.

O Obliq usa a idéia de agente como um programa que pode ser executado em um site remoto e agir como representante dos interesses do seu site de origem.

No modelo de objetos do Obliq não há o conceito de hierarquia; um objeto é auto-contido, o que, segundo o autor, é particularmente interessante para aplicações distribuídas, pois quando um objeto é transmitido pela rede ele pode trabalhar autonomamente no site destino, sem dependência da realocação de outros objetos da sua hierarquia.

A linguagem oferece suporte básico para segurança, assegurando que agentes provenientes de outros sites são impedidos de realizar operações "perigosas" no site atual. Este suporte é fornecido pela combinação de características que foram introduzidas para dar suporte a agentes distribuídos. O fato da linguagem ser interpretada em conjunto com os conceitos de escopo léxico e tipagem forte em tempo de execução são responsáveis por esta segurança. Devido ao escopo léxico agentes estrangeiros (provenientes de outros sites) têm acesso apenas aos dados e recursos do seu site original, os demais estão fora do seu escopo léxico. Para ter acesso a recurso do site atual, este deve ser explicitamente concedido pelo servidor deste site. Assim, existe um controle do que o agente pode realizar no site atual.

O conceito de Escopo Léxico é discutível, pois apesar de facilitar a distribuição dos dados e o tratamento do aspecto de segurança introduz bastante tráfego na rede.

A linguagem Obliq possui características semelhantes às de linguagens para programação distribuída, como, por exemplo, as da linguagem Orca [1] (uma linguagem para programação distribuída baseada em objetos).

O trabalho descrito em [5] descreve a integração da linguagem Obliq com browsers de web. São citados alguns browsers de web que foram desenvolvidos para aceitar objetos ativos escritos em Obliq - Oblets (*Ob*liq app*lets*). O trabalho com objetos ativos foi originalmente proposto pela linguagem Java (discutida na seção seguinte) e é o principal responsável pelo seu sucesso.

4.8.5. Java [13] [18] [19] [21]

A linguagem java é uma linguagem interpretada, projetada para ser usada em um ambiente distribuído, oferecendo suporte para execução de aplicações em diferentes ambientes de redes, múltiplos sistemas operacionais e arquitetura de hardware.

A inovação da linguagem é o trabalho com objetos ativos distribuídos (*applets*), que possibilita um browser carregar um programa remoto, executá-lo e mostrá-lo em uma página de web. Desta forma, a principal característica que diferencia java de outras linguagens é o suporte à transmissão de código que poderá ser executado em uma página de web. Java é usada para embutir conteúdos executáveis, também chamados de conteúdos interativos, em páginas de web.

Java não oferece suporte para programação distribuída, ela apenas distribui o seu interpretador. Não há como acessar um objeto via rede. Apenas código pode ser facilmente transmitido de acordo com o modelo java, o que significa que os dados devem ser comunicados via uma forma de comunicação convencional estilo socket.

O relacionamento de java com o paradigma de agentes está no suporte à agentes móveis. Os agentes móveis na linguagem java são chamados de *applets*. Os applets são programas que podem ser transmitidos via rede e precisam de um browser para serem executados. O applet representa o conteúdo interativo que pode ser carregado dentro de uma página de web.

4.8.5.1 Qual protocolo é utilizado para a comunicação entre os agentes?

Os applets apenas podem se comunicar com outros applets que estejam na mesma página de web. Não há uma maneira de um applet enviar informação para outro applet em outra página.

A comunicação entre os applets em uma mesma página é feita através da chamada direta de um método no applet destino.

4.8.5.2 Qual a linguagem é usada para programação de agentes?

Os applets são programados em java. A linguagem não oferece suporte para interação com applets escritos em outra linguagem.

4.8.5.3 O Controle da Distribuição é feito de que forma, ou seja, como é feita a localização dos objetos distribuídos?

A análise desta questão compreende a discussão de como os applets são incluídos em uma página de web e de como eles são localizados.

Para incluir um applet em uma página de web é adicionado um tag <applet> na página e quando esta página é acessada, o applet é automaticamente carregado do servidor origem pelo protocolo HTTP.

O <applet> deve referenciar um arquivo que não é parte da página HTML onde ele está embutido. Isto é feito através do parâmetro CODE, que identifica para o browser aonde procurar o arquivo .class.

Por exemplo, para introduzir o applet exemplo.class em uma página HTML, o seguinte arquivo HTML deve ser criado:

```
<HTML>
<HEAD>
<TITLE>  Applet Exemplo  </TITLE>
</HEAD>

<BODY>
Este é o applet: <P>
<APPLET codebase="classes"  code = "http://www.inf.puc-rio.br/~thais/exemplo.class"  width = 100
height= 100>
</BODY>
</HTML>
```

O applet é carregado automaticamente quando a página que contém é mostrada por um browser que seja capaz de exibir java. Este processo é similar à maneira pela qual o browser processa e mostra outros elementos como imagens e textos. A grande diferença com o applet é que o browser processa "conteúdo executado dinamicamente", ao invés de dados estáticos.

A localização do applet é feita pelo protocolo http através da URL que o programador explicitamente adiciona no parâmetro code. A URL contém o endereço de localização do applet.

Desta forma, java não oferece flexibilidade em relação à localização de objetos distribuídos. Não existe intermediários, nem passivo como na linguagem Obliq, nem ativo como os Facilitadores presentes na proposta de Genesereth, do KQML e do ORB do CORBA. Assim, o programador tem a responsabilidade de identificar explicitamente através da URL o applet desejado.

4.8.5.4 Quais termos dentro da linguagem são utilizados para garantir que outros agentes irão interpretar as expressões da mesma forma?

Devido à restrição de applets só se comunicarem com applets que estão na mesma página de web e também de apenas applets escritos em java poderem se comunicar, não há um suporte específico para a interpretação da comunicação entre applets. O programador que implementa esta comunicação é responsável também pelo controle da interpretação.

4.8.5.5 Qual a arquitetura seguida pelo sistema?

A linguagem java tem como base a plataforma java que possibilita a execução de applets e aplicações em um ambiente de redes de computadores.

A plataforma java situa-se no topo de outras plataformas e compila código java para *bytecodes*, que são instruções de máquina para uma máquina virtual, ou seja, não são específicos a qualquer máquina física. O *bytecode* pode ser executado em qualquer sistema operacional onde a plataforma java estiver presente.

A plataforma java tem duas partes básicas:

- Máquina Virtual Java: é a chave para a independência do sistema operacional subjacente e do hardware. A máquina virtual define um formato independente de máquina para arquivos binários, que inclui instruções para uma máquina virtual na forma de código de bytes (bytecodes).
- Interface de Programação de Aplicações Java (Java API): forma uma interface padrão para applets e aplicações independente do sistema operacional subjacente. Esta API especifica um conjunto de interfaces que auxilia o desenvolvimento de aplicações java.

Combinadas, estas partes oferecem um ambiente para desenvolvimento de aplicações para a Internet.

O desenvolvedor escreve código na linguagem java (arquivo .java) e compila para bytecodes (arquivo .class). Estes bytecodes são instruções para a máquina virtual java. Para criar um applet, o desenvolvedor armazena estes arquivos de bytecodes em um servidor HTTP e adiciona um tag <applet code = filename> em uma página de Web, cujos nomes são os pontos de entrada em arquivos bytecode.

Quando o usuário visita esta página, o tag <applet> causa o transporte do arquivo bytecode pela rede do servidor para o browser do usuário na plataforma Java. Depois, o bytecode é carregado na memória e tem aspectos de segurança verificados antes de entrar na máquina virtual.

Uma vez na máquina virtual, os bytecodes são interpretados pelo Interpretador, ou opcionalmente transformado em código de máquina pelo gerador de código just-in-time (JIT), mais conhecido como Compilador JIT. O interpretador e o compilador JIT operam no contexto do runtime system (*threads*, memória, outros recursos do sistema). Qualquer classe da biblioteca de classes Java (API) é dinamicamente carregada quando o applet necessita.

A figura 15 ilustra a arquitetura java que permite a transferência e execução de um applet em uma máquina remota.

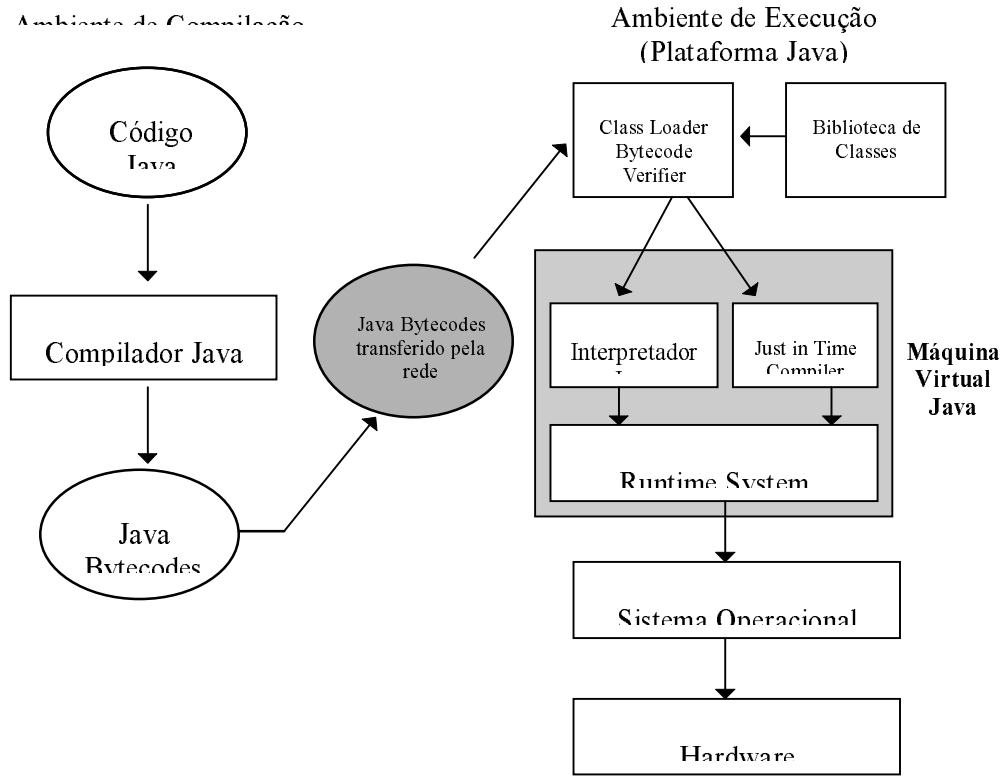


Figura 15 - Modelo de Execução Remota de código Java

4.8.5.6 Comentários

Java se tornou quase um padrão para construção de applets e aplicações portáveis via Internet porque permite aplicações serem escritas uma vez e executarem em uma variedade de sistemas operacionais e de plataformas de hardware. Esta portabilidade é garantida pela plataforma java, mais especificamente pela máquina virtual java e pela java API. Como as APIs ficam na plataforma java, então seguramente as aplicações podem executar em qualquer lugar que contenha esta plataforma.

Outro aspecto importante da linguagem java é o fato de ser interpretada. Uma linguagem sendo interpretada significa que ela pode ser independente da plataforma e portanto permitir a migração de programas e objetos para várias localizações.

Um aspecto crítico de java é que ela oferece independência de plataforma, mas não independência de linguagem. Assim, a interoperabilidade é restrita a applets escritos em java.

Uma das áreas com tratamento especial na linguagem java é Segurança. Para garantir que applets não realizem ações ilegais, quando o interpretador carrega *bytecodes* pela rede, eles são analisados por um processo de verificação. Além disto, existe uma API chamada **Java Security API** que é um esquema para programadores facilmente e seguramente incluirem segurança nos seus applets e aplicações. Esta funcionalidade inclui criptografia, assinatura digital, autenticação, gerenciamento de chaves e certificados.

Em relação à tecnologia de componentes, em [18] é ressaltado que os applets são componentes no sentido de que eles são objetos remotos que podem ser dinamicamente carregados em páginas de web. Porém, os applets constituem um modelo de componente bastante simples pois não podem interagir com outros applets que estejam em alguma localização arbitrária.

De forma a estender a capacidade da plataforma java com a finalidade de dar suporte à aplicações baseadas em componentes, a JavaSoft está desenvolvendo o JavaBeans [18] que é um Modelo de Desenvolvimento de Aplicações através da junção de componentes.

Muitas críticas [19] têm sido feitas à introdução do JavaBeans pois não foram emitidas publicações esclarecedoras de aspectos da sua especificação e desenvolvimento.

Um dos pontos ressaltados na única publicação disponível sobre o JavaBeans se refere à estratégia do JavaBeans para o uso de componentes em um ambiente distribuído. APIs do modelo de componentes do JavaBeans permite componentes interagirem em uma máquina virtual única. A justificativa é que ao invés de complicar a API ou a plataforma java com mecanismos robustos de computação distribuída, programadores java poderão escolher entre várias abordagens da computação distribuída. Por exemplo, será possível que os programadores implementem a interação com objetos distribuídos nas suas aplicações java usando o Método de Invocação Remota de java ou usando interfaces CORBA IDL para acesso a objetos remotos, ou ainda usando qualquer outro mecanismo de computação distribuída. Detalhes de como isto efetivamente funciona ainda não foram esclarecidos pelos desenvolvedores.

O trabalho brevemente descrito em [28] apresenta uma abordagem para criação de objetos remotos e invocação de métodos remotos em java através da introdução de novos construtores na linguagem.

5. Conclusão

Segundo [6] acabou o confronto entre sistemas operacionais, entre linguagens e também entre aplicações, e nenhuma delas ganhou a guerra. A batalha agora é na área de objetos interoperáveis. A evolução da programação orientada a objetos originou objetos independentes de linguagens e o esforço agora é na busca de tecnologias para interoperabilidade entre estes objetos.

Neste trabalho foram analisadas propostas em duas correntes que endereçam aspectos de interoperabilidade em ambientes distribuídos: Agentes de Software e Componentes de Software.

A maioria das propostas analisadas de interoperabilidade através de agentes de software [Esquema para Interoperabilidade, KQML, Obliq, AgenTalk e Java] são linguagens para dar suporte a agentes que oferecem extensões das características presentes em linguagens orientadas a objetos integrando características para que os objetos sejam vistos como agentes distribuídos em comunicação. Segundo [25], agentes distribuídos podem ser considerados como uma extensão de RPC com funções de migração. Eles são mais flexíveis que RPC porque eles são programáveis e tentam conservar banda passante mantendo o processamento mais próximo dos dados. Eles também são mais simétricos que outras formas de computação cliente/servidor baseadas em RPC: é possível para o servidor enviar código para um cliente bem como o cliente enviar código ao servidor (migração de processos).

Na área de componentes de software foram comentados aspectos pertinentes a este conceito e apresentado o padrão CORBA.

Em suma, pode-se concluir que um lado tenta resolver o problema da interoperabilidade desenvolvendo novas linguagens de programação [Obliq, AgenTalk, Java, KQML] facilmente tornadas disponíveis através de um interpretador, enquanto que o outro lado tenta estabelecer padrões que dão suporte a objetos de diferentes linguagens e oferece propostas que se situam no ponto intermediário entre o sistema operacional e a camada de aplicação [CORBA, DCE, OLE]. Nos sistemas que seguem a primeira filosofia foram analisadas linguagens que oferecem aspectos para permitir interoperabilidade entre objetos nela criados [Obliq, AgenTalk] e outras oferecendo suporte a objetos criados em linguagens diferentes [KQML]. Este ponto pode ser um passo em direção à desejável interoperabilidade entre objetos de diferentes origens, mas ainda não é o definitivo pois restringe muito a interoperabilidade ao uso específico de uma linguagem. A classe dos padrões é mais genérica, permitindo o compartilhamento de objetos de diferentes linguagens através do uso de uma interface padrão e de um barramento que disponibiliza componentes de software que seguem esta interface, permitindo assim o uso de linguagens compiladas e conseqüente verificação de tipos.

Nenhuma das duas propostas parece ser a definitiva, as pesquisas dão indícios que deve haver uma integração entre as duas tecnologias no sentido de somar esforços para oferecer uma solução abrangente para a interoperabilidade em ambientes distribuídos. A mobilidade do agente, possibilitando a proximidade de código e dados, é que permite esta abordagem ser usada para sistemas distribuídos. Como agentes são implementados em software é necessário uma maneira através da qual objetos de software possam migrar de uma máquina para outra. Desta forma, precisa-se das capacidades oferecidas pelo gerenciamento de objetos distribuídos baseados em padrões emergentes como o CORBA. Estes esquemas oferecem facilidades para migração de objetos e assim representam uma tecnologia adequada para o desenvolvimento de agentes de software em ambientes distribuídos [2]. Além de tudo, agentes de software são mais que coleções de estruturas de dados e métodos sobre elas. Assim, os padrões podem ser vistos como a base sobre a qual sistemas de agentes devem ser construídos.

Bibliografia

[1] Bal, H; Kaashoek, M.F.; & Tanembaum, A. "Orca: a language for parallel programming of distributed systems". IEEE Transactions on Software Engineering, 18( ), 1992, p.190-205.

[2] Beale, R. & Wood, A. "Agent-Based Interaction". Proceedings of HCI'94, Glasgow, UK, August 1994, p.2 9-245.

[ ] Belgrave, M. "Topics in Agent-Oriented Programming: Intelligent, Autonomous Agents and Prolog". McGill University, Montreal. http://www.ee.mcgill.ca/~belmarc/

[4] Bernstein, P. "Middleware". Communications of the ACM, Vol. 9,N° 2, February 1996.

[5] Brown, M. & Najork, M. "Distributed Active Objects". Computer Networks and ISDN Systems, 28, 1996, p. 10 -1052.

[6] Buarque de Holanda, A. *Dicionário Aurélio*. Editora Nova Fronteira, 1986.

[ ] Cardelli, L. "A Language with Distributed Scope" Report N° 122. Digital Equipment Corporation, Systems Research Center. 1994

[8] Cerqueira, R. *Um Estudo sobre Interoperabilidade entre Linguagens Orientadas a Objetos*. Dissertação de Mestrado. Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro. Março 1996.

[9] Chalupsky, H. et al. An Overview of KQML: A Knowledge Query and Manipulation Language. http://www.cs.umbc.edu/agents/technology/ April 1995.

[10] Comer, D. & Stevens, D. *Internetworking with TCP/IP - volume III*, Prentice-Hall Editions, 199 .

[11] Ellman, T. "Explanation-based learning: A survey of programs and perspectives". ACM Computing Surveys. 21, 2, 1989, p. 164-221.

[12] Fingar, P. & Stikeleather, J. "Distributed objects for business". http://www.sun.com/sunworldonline/swol-04-1996/swol-04-oobook.html

[l] Friedel Jr., D. & Potts, A. Java Programming Language Handbook. 1996

[14] Genesereth, M. & Ketchpel, S. "Software Agents". Communications of the ACM. Vol. , No. , July 1994.

[15] Genesereth, M. "Interoperability: An Agent-Based Framework". AI Expert, March 95.

[16] Harrison, C.; Chess, D. & Kershenbaum, A. "Mobile Agents: Are they a good idea?" IBM T. J. Watson Research Center. Yorktown Heights, NY, March 1995.

[l] Heuser, L et al. "Development of Distributed and Client/Server Object-Oriented Applications: Industry Solutions", ACM Sigplan Notices, Vol.29, N°10, October 1994.

[18] Java(tm) Beans: A Component Architecture for Java. http://splash.javasoft.com/beans/WhitePaper.html

[19] Java Forum. http://www.javasoft.com/forum/

[20] Kozierok, R. & Maes, P. "A learning interface agent for scheduling meetings." ACM-SIGCHI International Workshop on Intelligent User Interfaces. Orlando. January 9 , 1-8.

[21] Kramer, D. "The Java Platform", http://java.sun.com/doc/WhitePaper.Platform/CreditsPage.doc.html, May 1996.

[22] Kuwabara, K.; Ishida, T. & Osato, N. "AgenTalk: Describing Multiagent Coordinaton Protocols with Inheritance". Proc. th International Conference on Tools with Artificial Intelligence (ICTAI'95) , 1995, pp. 460-465.

[2] Lewine, D. *Posix Programmer's Guide*. O/Reilly & Associates, Inc. 1991.

[24] Mayfield, J. et al. "Evaluation of KQML as an Agent Communication Language". http://www.cs.umbc.edu/agents/technology/

[25] Majewski, S. "Distributed Programming: Agentware/Componentware/Distributed Objects". http://minsky.med.virginia.edu/sdm g/Projects/Python/SafePython.html

[26] Martin, J. *SNA: IBM's Networking Solution*, Prentice-Hall, 198 .

[2] Murray, D. "Developing Reactive Software Agents", AI Expert March 95.

[28] Nagaratnam, N. & Srinivasan, A. "Remote Objects in Java". http://www.cat.syr.edu/~nataraj/remoteJavaObjs.ps.gz

[29] Neufeld, G. & Vuong, S. "An Overview of ASN.1". Computer Networks and ISDN Systems, Vol. 2 , 1992, pp. 9 -415.

[0] Nierstrasz, O.; Gibbs, S.; Tsichritzis, D. "Component-Oriented Software Development". Communications of the ACM, Vol. 5, No.9, September 1992, pp. 160-165.

[1] Object Management Group. "The Common Object Request Broker: Architecture and Specification (Revision 2.0)". Object Management Group (OMG), Framingham, MA., July 1995.

[2] Orfali, R.; Harkey, D. & Edwards, J. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., 1996.

[ ] Sánchez, J.; Leggett, J. & Schnase, J. "HyperActive: Extending an Open Hypermedia Architecture to Support Agency". ACM Transactions on Computer-Human Interaction (TOCHI) Vol. 1, No. 4, December 1994.

[4] Tanembaum, A. *Computer Networks*. Prentice-Hall Editions, 1989.

[5] Thirunavukkaray, C.; Finin, T. & Mayfield, J. "Secret agents - a security architecture for the KQML agent communication Language". In Proceedings of the ACM CIKM Intelligent Information Agents Workshop, December 1995.

[6] Valdés, R. "Introducing Interoperable Objects". Dr. Dobb's Special Report: The Interoperable Objects Revolution. Vol. 19. Winter 1994/1995.

[ ] Wayner, P. "Agents of Change", Byte. March 1995.

[8] Yang, Z. & Duddy, K. "CORBA: A Platform for Distributed Object Computing", Operating Systems Review, N° 2, Vol. 0, April 1996.