Um Modelo de Negociação Automatizada para Comércio Eletrônico

Jó Ueyama

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Um Modelo de Negociação Automatizada para Comércio Eletrônico

Jó Ueyama

Julho de 2001

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira
  Instituto de Computação - Unicamp (Orientador)
- Prof. Dr. Antonio Alfredo Ferreira Loureiro
  Departamento de Ciência da Computação - UFMG
- Prof. Dr. Rogério Drummond Burnier Pessoa de Mello Filho
  Instituto de Computação - Unicamp
- Prof. Dr. Jorge Stolfi
  Instituto de Computação - Unicamp (Suplente)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Ueyama, Jó

Ue9m Um modelo de negociação automatizada para comércio eletrônico / Jó Ueyama -- Campinas, [S.P. :s.n.], 2001.

Orientador : Edmundo Roberto Mauro Madeira

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

1. Negociação. 2. Agentes inteligentes (Software). 3. Sistemas de computação. I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Um Modelo de Negociação Automatizada para Comércio Eletrônico

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Jó Ueyama e aprovada pela Banca Examinadora.

Campinas, 30 de Julho de 2001.

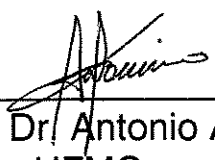
Edmundo R M Madeira

Prof. Dr. Edmundo Roberto Mauro Madeira
Instituto de Computação - Unicamp
(Orientador)

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

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 06 de julho de 2001, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Antonio Alfredo Ferreira Loureiro
DCC - UFMG

Prof. Dr. Rogério Drummond Burnier Pessoa de Mello Filho
IC - UNICAMP

Prof. Dr. Edmundo Roberto Mauro Madeira
IC – UNICAMP

Resumo

Existem poucos modelos de negociação bilateral implementados para aplicações de comércio eletrônico que provêem a negociação entre o comprador e o vendedor. Esta dissertação propõe um protocolo de negociação entre dois participantes e um mecanismo para medir as similaridades entre dois produtos, utilizado para obter o produto mais similar ao que foi requisitado pelo consumidor quando um exato não for encontrado. O protocolo proposto segue o modelo bilateral do *OMG*. A negociação é tratada pelos agentes móveis (*Grasshopper*) de compra e de venda, representando respectivamente o comprador e o vendedor. Por sua vez, a negociação do preço é baseada no modelo *Kasbah*. Os catálogos no modelo foram implementados em *XML* para prover a interoperabilidade entre diferentes arquiteturas. Apresentamos resultados de diversas simulações com o uso de catálogos em XML. Um protótipo foi desenvolvido para validar o modelo proposto.
Palavras-chave: E-Commerce, Negociação Automatizada, Agentes Móveis.

Abstract

Electronic commerce applications are lacking a bilateral negotiation model which provides the bargaining between two participants (supplier and consumer) in order to buy and sell goods. This dissertation presents a negotiation protocol between two participants, as well as the *similarity measures* which were implemented to find a similar product, when a specific one could not be found. The proposed protocol follows the bilateral model approved by the *OMG*. The negotiation model is composed of selling and buying *Grasshopper* mobile agents which negotiate between them in order to get the best deal. The negotiation of the price is based on the *Kasbah* model. The catalogs in the model are implemented in *XML* to provide the interoperability among different systems. We present several simulation results regarding the use of XML based catalogs. A simple prototype has been implemented to verify the viability of this model.
Keywords: E-Commerce, Automated Negotiation, Mobile Agents

Agradecimentos

Agradeço...

À Deus, a quem recorri em tantas ocasiões e por inúmeras razões, agradeço pelas inspirações, pela força, pela disposição, pela perseverança, pela saúde e, principalmente, por ter me iluminado quando me achava perdido entre dúvidas e incertezas. Obrigado pelos desafios e pela coragem de enfrentá-los.

Aos meus pais (Yoshimi e Taiko), pelo amor, dedicação, carinho e, sobretudo, pelo apoio e incentivo que sempre me ofereceram, respeitando minhas decisões de vida mesmo quando resultaram em caminhos por demais tortuosos para todos nós.

Às minhas irmãs, Mary, Monica e Feliciana pela ajuda, pelo pensamento positivo e pelos estímulos constantes.

Ao meu orientador, Edmundo R. M. Madeira, pela sua valiosa orientação e sobretudo pelo seu profissionalismo e dedicação durante o desenvolvimento desta dissertação. Agradeço mais uma vez a Deus pelo privilégio de tê-lo como meu orientador.

Ao pastor Fernando Garcia Leite, pelo apoio durante a minha estadia aqui em Campinas, além dos valiosos ensinamentos ministrados.

Aos meus amigos e companheiros de república, Delano, Marcos André, Alexandre e Eduardo, por terem propiciado a criação de um ambiente amigável durante os vários meses de convivência.

À Márcio de Oliveira Buss e Tallys Hoover Yunes pelas idéias sugeridas e implementadas neste trabalho

Aos meus amigos do laboratório do DCA (Departamento de Computação e Automação)

pelo apoio recebido durante a fase da implementação desta dissertação.

Ao meu amigo Marco Antonio Rossi, pela amizade e companheirismo durante a minha estadia aqui em Campinas.

Aos professores e aos funcionários do Instituto de Computação que direta ou indiretamente contribuíram ainda mais para minha formação profissional.

Aos amigos e colegas do IC que de diversas formas me apoiaram e proporcionaram momentos alegres, além do convívio, dos conselhos e da troca de experiências. Agradeço muito à Deus por tê-los conhecido.

Ao Prof. Jorge Stolfi pela disposição em ler os meus artigos e sugerir idéias novas no nosso trabalho.

À Assembléia Legislativa do Estado do Pará que proporcionou o apoio financeiro para viabilizar a confecção deste trabalho.

Sumário

Lista de Figuras

Capítulo 1

Introdução

O comércio eletrônico tem se mostrado através do crescimento comprovado pelas estatísticas, como a nova e promissora forma de comércio, dentro do contexto da economia global. Porém apesar destas estatísticas otimistas, as aplicações de comércio eletrônico necessitam ainda de muitas melhorias, para atender de forma adequada o usuário que utiliza desse mecanismo de comércio. A grande dificuldade para os usuários da Internet, como também do comércio eletrônico, é a falta de uma estrutura padronizada que permita procurar e encontrar as informações de forma simples e otimizada.

A falta de uma padronização nos sistemas de comércio eletrônico dificulta a implementação da interoperabilidade nos mesmos. Tendo em vista que esses sistemas utilizam a Internet para operacionalizar as suas funcionalidades, fica claro que a interoperabilidade nos sistemas de comércio eletrônico é uma questão vital para que o e-commerce possa sair do seu estado embrionário, onde se encontra atualmente. Outra questão em aberto é a implementação de mecanismos eficazes de negociação nesses sistemas. Como o comércio eletrônico envolve a troca de bens, é natural que haja uma negociação entre os participantes, para que se chegue a um acordo entre as partes envolvidas. Apesar da presença maciça de leilões na Internet, mecanismos eficazes de negociação entre consumidor e vendedor ainda não são largamente utilizados.

O propósito principal deste trabalho é apresentar um modelo de negociação automatizada entre o consumidor e diversos vendedores. Além de abordar o modelo proposto, este trabalho discute o uso de XML (*Extensible Markup Language*) [40] na implementação de catálogos eletrônicos, citando as vantagens, além de apresentar vários resultados obtidos a partir de simulações efetuadas com os catálogos em XML. Neste trabalho apresentamos várias simulações com a finalidade de testar o uso de RMI (*Remote Method Invocation*), CORBA (*Common Object Request Broker Architecture*) e agentes móveis (*Grasshopper*) para acessar catálogos em XML. Vale ressaltar que os catálogos nesse modelo são projetados em XML com o intuito de prover a interoperabilidade entre arquiteturas diferentes.

Um protocolo de negociação bilateral (dois participantes) é proposto para que consumidor e fornecedor possam negociar entre eles. Este modelo foi implementado em *Java* utilizando o sistema de agentes móveis *Grasshopper*. Além do protocolo, propomos uma medida de similaridade que tem a finalidade de determinar as semelhanças entre os produtos. Esta abordagem é de extrema importância em virtude da necessidade de se determinar o produto mais similar quando a mercadoria exata não for encontrado pelo fornecedor. Caso o produto requisitado pelo consumidor não esteja disponível no momento, o modelo é capaz de ofertar um que seja similar, utilizando a abordagem da similaridade proposta.

As funcionalidades básicas de negociação utilizadas neste trabalho são baseadas na Facilidade de Negociação do OMG (*Object Management Group*). O nosso protocolo utiliza também algumas funcionalidades do modelo de *Kasbah* [6] e do *Contract Net Protocol* [35]. Assim como no *Kasbah*, o nosso protocolo é baseado nos agentes móveis de venda e de compra que representam respectivamente, o comportamento do vendedor e do comprador. Da mesma forma que no protocolo *Contract Net*, o nosso protocolo implementa também a chamada por propostas.

A presente dissertação apresenta as seguintes contribuições:

- Protocolo de negociação automatizada que permite a barganha paralela e autônoma com diferentes fornecedores de forma simultânea;
- Medidas de similaridades (mecanismo para determinar os produtos similares) que não são baseados em técnicas de aprendizado;
- Simulação com catálogos em XML em ambientes locais e remotos, realizando comparações com as três tecnologias de comunicação remota mais comumente utilizadas (RMI, CORBA e agentes móveis *Grasshopper*);
- Comparação entre as duas APIs mais comumente utilizadas no mercado para integrar XML e Java.

1.1 Organização da Dissertação

- O Capítulo 2 introduz alguns conceitos de comércio eletrônico, catálogos e as tecnologias de comunicação remota utilizadas na dissertação. O referido capítulo aborda também a facilidade de negociação especificada pelo OMG.
- Os aspectos relacionados aos catálogos em XML são discutidos no Capítulo 3. Neste capítulo apresentamos as vantagens dos catálogos em XML além dos resultados obtidos a partir das simulações executadas com estes catálogos.

- As questões relacionadas à negociação são abordadas no Capítulo 4. O referido capítulo discute a negociação automatizada assim como os três modelos de negociação do OMG: Bilateral (negociação entre dois participantes), Multilateral (negociação entre vários participantes) e o Modelo de Promissória (negociação para liquidar uma promissória).
- O protocolo de negociação e o mecanismo proposto para definir as similaridades entre os produtos são apresentados no Capítulo 5.
- O Capítulo 6 é destinado para discutir aspectos relacionados à implementação do modelo.
- O Capítulo 7 conclui a dissertação tecendo alguns comentários relevantes do nosso trabalho e propondo extensões.

Capítulo 2

Conceitos Básicos

Neste Capítulo apresentamos os conceitos básicos de comércio eletrônico e tecnologias de comunicação remota (RMI, CORBA e agentes móveis *Grasshopper*). Abordamos também a arquitetura proposta pelo OMG e CommerceNet para aplicações de comércio eletrônico baseadas em CORBA. Cada componente desta arquitetura é apresentado com a finalidade de fornecer subsídios aos capítulos posteriores, porém é válido salientar que o enfoque maior será dado aos componentes que serão utilizados pelo nosso modelo. Maiores detalhes referentes à arquitetura da CommerceNet podem ser encontrados em [9].

2.1 Comércio Eletrônico

O comércio eletrônico é uma nova forma de comércio, onde o produto é conhecido, demonstrado e vendido por meios eletrônicos. Atualmente o meio mais popular é a Internet. Atualmente, as aplicações de comércio eletrônico compreendem a propaganda, catálogos, pagamento de contas, assim como a compra e venda de bens/serviços. A localização geográfica para que esses serviços sejam operacionalizados é irrelevante, posto que a Internet tem presença mundial, de forma a contribuir sensivelmente para a globalização do comércio mundial.

Os negócios realizados atualmente são caracterizados pela modernização das tecnologias dos fornecedores assim como pela crescente exigência por parte dos consumidores. Em resposta a esse contexto, os negócios realizados atualmente estão mudando de comportamento, que vai desde a mudança na sua organização interna, até o modo como cada negócio é operacionalizado. Hoje em dia, a organização hierárquica interna da empresa está saindo de uma característica vertical para uma estrutura horizontal, eliminado assim cada vez mais as camadas existentes entre clientes e fornecedores [1].

O comércio eletrônico é considerado como o meio que suporta as mudanças consideradas anteriormente, permitindo que as companhias sejam mais eficientes e flexíveis,

para que assim possam agir mais próximo de seus fornecedores e ao mesmo tempo mais sensíveis às expectativas de seus clientes [1].

2.1.1 Categorias

O comércio eletrônico pode ser dividido em quatro categorias distintas, quanto às entidades participantes [12]:

- **Empresa-Empresa**: Quando as empresas se comunicam usando computadores para fazer pedidos, receber faturas ou realizar pagamentos. Esta categoria de comércio eletrônico tem sido utilizada através da EDI (*Electronic Data Interchange*), em redes proprietárias.
- **Empresa-Governo**: Engloba toda a interação entre empresa e governo via computadores. Nos EUA, a Internet já é usada em concorrências públicas, e no Brasil para a entrega das declarações do Imposto de Renda. Esta categoria ainda se encontra incipiente, mas deve se desenvolver rapidamente.
- **Empresa-Consumidor**: Corresponde ao varejo eletrônico. Esta categoria encontra-se em expansão graças à WWW.
- **Consumidor-Governo**: Começa a ser uma alternativa para que os cidadãos possam manter em dia suas declarações de impostos, entre outras atribuições. Bons exemplos são as declarações de impostos de 1997 a 2001, que puderam ser entregues pela Internet.

2.1.2 Vantagens

O comércio eletrônico possui diversas vantagens em comparação com o comércio face-a-face tradicional [12]:

- Diminui o tempo e custo de busca, tanto para os clientes quanto para os fornecedores;
- Expande mercados locais e regionais para nacionais e internacionais, com níveis mínimos de capital, estoque e pessoal;
- Decrementa os altos custos envolvidos em transporte, armazenamento e distribuição, bem como em identificar e negociar com potenciais clientes e fornecedores;

2.1.3 Requisitos

Existem alguns serviços de infra-estrutura e elementos indispensáveis nas arquiteturas de sistemas de comércio eletrônico. Estes elementos possibilitam a flexibilidade, interoperabilidade e abertura nas implementações para que a tecnologia possa evoluir de forma consistente e estruturada. Alguns destes elementos são apresentados abaixo [30]:

- **Interoperabilidade**: Os sistemas de comércio eletrônico devem estar baseados em um conjunto comum de serviços e padrões que garantam a interoperabilidade. Dessa forma, provedores de serviços e desenvolvedores de aplicações poderão utilizar estruturas modulares que poderão ser combinadas, aperfeiçoadas e customizadas.

- **Flexibilidade para inovações**: Os produtos e serviços existentes serão completamente redefinidos e modificados. As soluções, apesar de sofisticadas, deverão possibilitar a execução de modificações de forma rápida e simples.

- **Oferta de *soft-products***: Produtos como publicações, catálogos, vídeos, programas, vídeo-games e até chaves eletrônicas de quartos de hotel, carros, entre outros, não vão ser simplesmente oferecidos aos clientes. Eles serão *especificados* pelos clientes, que poderão, por exemplo, *montar* um CD com as músicas de sua preferência. Este tipo de recurso agrega ao processo de venda uma fase de projeto por parte do cliente.

- **Novos Métodos de Receita**: O comércio eletrônico deverá suportar formas alternativas de coleta de receita, como pagamento contra recibo, pagamento adiantado, entre os diversos existentes. Outra mudança deverá ocorrer com relação à comercialização de software de forma geral (programas, jogos, vídeo e áudio). O provedor de serviço poderá cobrar a utilização do produto por demanda.

- **Integração aos sistemas legados**: Muitos sistemas já existentes nas organizações deverão interagir com as soluções de comércio eletrônico. Como estes sistemas não serão substituídos da noite para o dia, as soluções deverão permitir que sejam totalmente transparentes para o usuário na obtenção de informações que provenham dos sistemas legados.

2.2 Catálogos

O Catálogo de produtos é um conjunto de informações que descrevem um determinado produto e/ou serviço, servindo como principal elo de ligação entre o consumidor e o fornecedor. Assim como definimos os campos e os registros de um determinado arquivo

para descrever o seu conteúdo, o catálogo também é estruturado de forma a descrever o produto e/ou serviço que está sendo armazenado, através de um conjunto de atributos.

Os catálogos de produtos no ambiente de comércio eletrônico possuem dois tipos de restrições que valem a pena serem considerados: restrição temporal e a restrição espacial. A primeira define a vigência do catálogo quanto ao tempo. Por exemplo, existem catálogos que possuem certos valores durante o dia e outros valores durante a noite. A restrição espacial define as questões relacionados ao espaço. Por exemplo, alguns produtos podem possuir preços diferentes dependendo do local onde o consumidor estiver adquirindo o produto.

A Figura 2.1 ilustra o catálogo de produtos e/ou serviços entre o fornecedor e o consumidor.

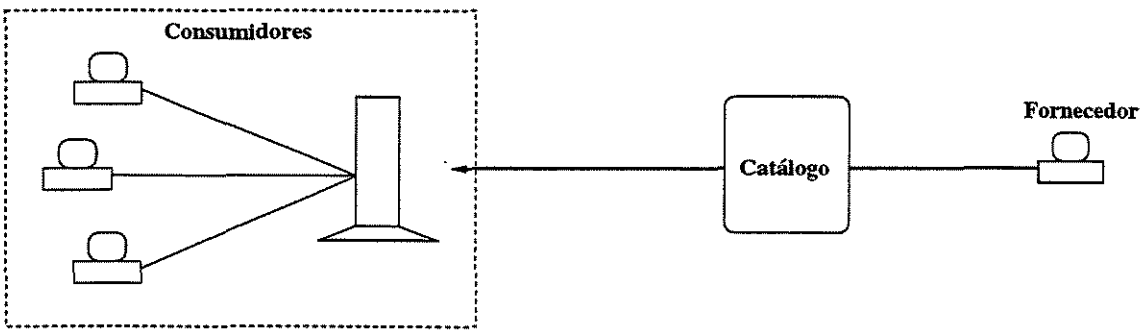


Figura 2.1: Catálogo em comércio eletrônico

2.3 Tecnologias de Comunicação Remota

2.3.1 RMI

RMI (*Remote Method Invocation*) permite que um objeto escrito em Java, rodando sobre um JVM (*Java Virtual Machine*), chame um método de um outro objeto Java que esteja rodando. RMI permite a comunicação remota entre programas escritos em Java. A grande vantagem do RMI é o fato de que o objeto inteiro pode ser passado e retornado como parâmetro, diferente dos outros mecanismos de chamada remota. Essa propriedade é bastante interessante, uma vez que ela permite que os códigos sejam transportados e carregados dinamicamente em tempo de execução em uma outra máquina virtual remota.

2.3.2 Agentes Móveis *Grasshopper*

Agentes móveis são programas que podem se locomover fisicamente, transportando o código e os seus estados, através da rede para executar tarefas em benefício de uma pessoa. Essa propriedade de mobilidade permite que os processos migrem de um computador a outro, além de permitir que um único processo seja dividido em múltiplas instâncias com a finalidade de dividir a sua execução entre várias máquinas diferentes, retornando ou não ao local de origem, após o término da sua execução. Os agentes móveis diferem dos mecanismos de chamada remota no que diz respeito à mobilidade do código executável.

Existem várias plataformas de agentes móveis que foram considerados na nossa pesquisa. Dentre as diversas podemos citar: *Aglets* da IBM [14], *Voyager* da ObjectSpace [22] e *Grasshopper* da IKV [17]. A seguir vamos detalhar a plataforma *Grasshopper* que foi utilizada na implementação do protótipo.

Grasshopper é uma plataforma de agentes móveis desenvolvida pelo *IKV++* e lançado no mercado em agosto de 1998. Sua característica principal é a sua especificação que está totalmente compatível com a especificação MASIF (*Mobile Agent System Interoperability Facility*) definida pelo OMG. A especificação MASIF foi lançada para permitir a interoperabilidade entre os sistemas de agentes móveis de diferentes desenvolvedores. A escolha pela utilização do sistema de agentes *Grasshopper* foi justamente em virtude da mesma seguir a especificação MASIF que possibilita a abertura de uma sessão de negociação entre agentes móveis de plataformas diferentes.

Componentes da Plataforma

A plataforma *Grasshopper* é composta de *regiões, agências* e *lugares*. Cada um dos componentes é descrito a seguir[17]:

Agência: Uma *agência* é o local onde os agentes estacionários e móveis são executados. Pelo menos uma agência deve existir em cada *host* para que os agentes possam ser executados.

Região: *Região* é um conceito utilizado no *Grasshopper* com a finalidade de facilitar o gerenciamento dos componentes distribuídos (*agências, lugares* e *agentes*).

Lugar: *Lugar* permite um agrupamento lógico entre os agentes pertencentes a uma mesma *agência*. As *agências* juntamente com os *lugares* podem ser associados a uma determinada *região* permitindo um nível de agrupamento maior.

Serviço de Comunicação

A comunicação entre os agentes no *Grasshopper* é realizada através de diferentes formas. *Grasshopper* suporta os modelos de comunicação descritos a seguir:

Comunicação Síncrona: Neste modelo, o cliente invoca o método no servidor que por sua vez executa a chamada, retornando os resultados ao cliente que prossegue com o seu processamento após a recepção desses resultados. Esta comunicação é denominada como *síncrona*, uma vez que o cliente fica bloqueado até que os resultados sejam retornados a ele.

Comunicação Assíncrona: O Mecanismo *assíncrono* não requer que o cliente fique bloqueado até a recepção dos resultados, porém ele pode chamar o método e continuar a execução da próxima tarefa. Existem várias formas para que o cliente receba os resultados do método chamado: o cliente pode perguntar periodicamente ao servidor se a execução do método foi finalizado, aguardar até que o resultado seja requisitado no cliente, ou solicitar que notifique quando o resultado estiver disponível.

Comunicação Dinâmica: Nesta comunicação o cliente é capaz de construir a mensagem em tempo de execução, especificando a assinatura do método a ser invocado. Este mecanismo de comunicação é utilizado quando o cliente não está ciente das funcionalidades desempenhadas pelo método. Esta comunicação pode ser realizada de forma *assíncrona* ou *síncrona*.

Comunicação Multicast: Esta modalidade permite o paralelismo na interação com os objetos do servidor. Com este mecanismo, o cliente é capaz de invocar o mesmo método localizado nos diferentes servidores.

2.3.3 CORBA

CORBA é o projeto de *middleware* desenvolvido por um consórcio formado por mais de 800 entidades de todas as áreas da computação, chamado de OMG [29]. O trabalho deste grupo concentra-se no desenvolvimento da especificação da Arquitetura de Gerenciamento de Objetos - OMA (*Object Management Architecture*), dentro dela o componente mais importante é a CORBA. A arquitetura CORBA, além de suportar a interoperabilidade entre objetos de diferentes plataformas, provê vários serviços de manipulação de objetos (criação, acesso e exclusão), além de suportar o armazenamento em repositórios persistentes [29].

ORB (*Object Request Broker*) é o componente que permite que o cliente faça requisições de forma transparente a outros objetos que se encontram em repositórios locais ou

remotos. O ORB faz a função de um corretor, procurando pelos objetos que satisfaçam às requisições dos clientes. O cliente não necessita estar ciente dos mecanismos utilizados para comunicar com os objetos servidores, assim como para ativá-los.

A IDL (*Interface Definition Language*) é a linguagem que descreve as interfaces dos objetos. Uma interface é definida através das operações suportadas e dos parâmetros necessários para manipular o objeto. IDL é o mecanismo utilizado para definir as operações disponíveis, e como cada uma delas poderá ser invocada. Através da IDL é possível mapear objetos CORBA para uma linguagem de programação específica. A Figura 2.2 ilustra o mapeamento de cada objeto, através de sua IDL, para interoperar com outros objetos desenvolvidos em plataformas diferentes.

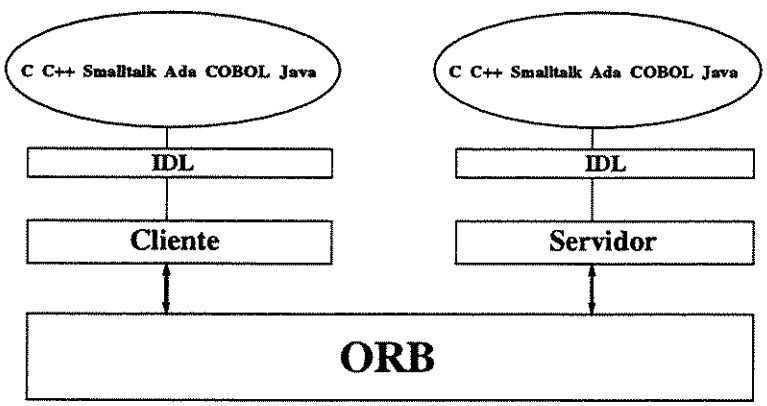


Figura 2.2: Interoperabilidade Cliente/Servidor através da IDL

Serviços CORBA formam uma coleção de serviços definidos através de interfaces IDL. O seu principal objetivo é ampliar as funcionalidades do ORB. Através dos mesmos é possível criar, nomear e introduzir os objetos no ambiente CORBA. Dentre os diversos serviços existentes, podemos enumerar o serviço de nome, ciclo de vida, persistência, evento e controle de concorrência. Um serviço que merece atenção para o nosso modelo é o serviço de *Trader* ou *Trading* [29] que fornece um serviço de "páginas amarelas". Os objetos anunciam suas ofertas de serviços ao *Trader*, que por sua vez pode ser consultado, para localizar pela oferta, com as características descritas pelo cliente. Se o serviço procurado estiver no seu domínio, a referência desta interface é retornada ao cliente, de forma que ele faça a sua requisição. Uma abordagem mais aprofundada em CORBA pode ser encontrada em [27, 29].

2.4 Arquitetura para Comércio Eletrônico Proposta pelo OMG e CommerceNet

A Figura 2.3 ilustra os componentes da arquitetura proposta pelo OMG em conjunto com a associação CommerceNet, apresentado num *Whitepaper* [9]. Esta arquitetura é dividida em três grupos principais: (1) **Serviços de Comércio Eletrônico de Baixo Nível** que inclui os serviços de pagamento, facilidade de dados semânticos e serviços de negociação/seleção; (2) **Facilidade de Comércio** suportando contratos, gerenciamento de serviços e facilidades relacionadas a *desktop*; e (3) **Facilidades da Infra-Estrutura de Mercado** que inclui catálogos, corretagem (*brokerage*) e agências [32].

Figura 2.3: Arquitetura de Comércio Eletrônico Proposta pelo OMG

2.4.1 Serviços de Comércio Eletrônico de Baixo Nível

Facilidade de Dados Semânticos

O requisito principal no comércio eletrônico é a estrutura comum para a troca de dados que descreva produtos, serviços, conteúdos e bens. O segundo requisito diz respeito à necessidade de se poder construir dinamicamente descrições de serviços que possam ser modificadas e estendidas. As alterações das descrições de serviços podem ser ocasionadas como resultado de uma negociação e seleção entre os participantes.

Dados semânticos podem ser considerados como objetos de dados, fornecidos por um provedor de serviços que descreva os serviços oferecidos ou requisitados. Eles podem ser considerados como recipientes portáteis e persistentes para objetos de dados de qualquer tipo.

O principal requisito para um SDO (*Semantic Data Object*) é a possibilidade de descrever qualquer serviço ofertado em um mercado eletrônico.

Dentre os diversos requisitos a serem atendidos pela Facilidade de Dados Semânticos, podemos citar:

1. Devem ser genéricos para incluir informação de qualquer tipo;
2. Possibilidade de se incluir, alterar ou excluir informação dentro de um SDO;
3. Para cada objeto de informação que se encontra dentro de um SDO, deve ser possível determinar seu tipo e sua estrutura;
4. Deve ser possível navegar dentro de um SDO por interação, assim como pela pesquisa por tipo e informação;
5. Conversão do SDO armazenado na memória principal para uma forma persistente e vice-versa;
6. A forma persistente de um SDO deve ser portável e independente de plataforma.

Além dos requisitos funcionais citados, existe um outro requisito que diz respeito à padronização dos rótulos no intuito de prover suporte à evolução dos mercados abertos, através da definição dos termos comuns entre os mercados.

No modelo proposto por este trabalho, a Facilidade de Dados Semânticos será implementada através do uso de XML para descrever os produtos existentes no catálogo.

Serviços de Seleção/Negociação

Os serviços de negociação suportam a seleção e a configuração das facilidades existentes nos diversos domínios que envolvem o ambiente onde as transações de comércio eletrônico estejam sendo executadas. Estas transações requerem uma política comum, sendo assim mecanismos para disponibilizar serviços e facilidades e a subsequente convergência para um comum acordo quanto às configurações desses serviços e facilidades se fazem necessários. Em suma, a facilidade de negociação é o intermediário entre os domínios diferentes, de forma que os mesmos possam oferecer serviços e facilidades convergindo-se para um comum acordo quanto às suas configurações.

As seguintes definições se aplicam para o contexto descrito acima:

- política macro: política que liga os domínios durante uma transação comercial;
- política de revelação: política que trata da divulgação das informações de um domínio local para outros externos.

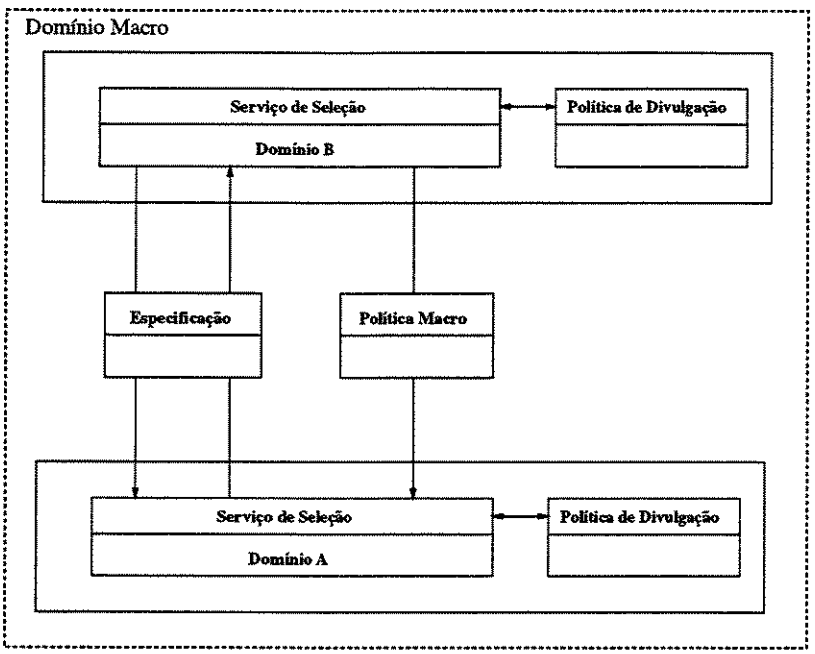


Figura 2.4: Serviço de Negociação - da especificação à macro-política

O nosso trabalho tem como objetivo principal propor um modelo de negociação automatizada que possa levar a um acordo entre os participantes. Vale salientar que o nosso modelo propõe a negociação bilateral (negociação envolvendo dois participantes).

Os requisitos do Serviço de Seleção incluem:

- habilidade de emitir e receber a especificação dos requisitos;
- permitir que vários participantes possam negociar sobre uma determinada especificação;
- suportar a flexibilidade quanto à divulgação das informações de acordo com um contrato pré-determinado;
- Acordo na negociação dos requisitos e compromisso para cumprí-lo.

Na Figura 2.4, o domínio B requisita informação do domínio A para que B possa utilizar algum serviço ou facilidade disponível em A. Os domínios A e B iniciam o processo de negociação até que se chegue a um comum acordo entre eles. Após o acordo entre ambos, é criada uma especificação, estabelecendo-se uma política macro entre os mesmos.

A divulgação de cada informação durante o processo de negociação é regida por uma política privada existente nos dois domínios.

O Serviço de Seleção deve ser suficientemente genérico de forma a tratar informações generalizadas. As informações tratadas estão contidas nas seguintes requisições:

- requisição sobre alguma facilidade disponível no domínio;
- requisição das informações relacionadas ao negócio;
- requisição das informações referentes à função e identidade;
- requisição nas informações relacionadas às agências do negócio;
- requisição das informações dos serviços comerciais.

Facilidade de Pagamento Eletrônico

A explosão do comércio eletrônico na Internet tem criado novos requisitos para pagamento eletrônico. Esses pagamentos podem ser efetuados de diversas formas: cartão de crédito, dinheiro eletrônico, micro-pagamentos, etc. Além disso, o pagamento pode ser efetuado através de entidades diferentes como bancos, consumidores e comerciantes, além de outras existentes. Para solucionar a questão do pagamento eletrônico, é necessário um sistema de comunicação seguro, confiável e que possa atender às necessidades do mercado.

Alguns protocolos de pagamento são específicos para atender a uma determinada área. Um exemplo disso é o SET (*Secure Electronic Transaction*) que foi padronizado para ser utilizado pelas administradoras de cartão de crédito.

2.4.2 Facilidade de Comércio

Facilidade de Gerenciamento de Serviço

Um serviço pode ser descrito como provisão de algo de um provedor para um consumidor. Exemplos de serviço no comércio eletrônico incluem: o pagamento eletrônico, onde o pagamento é efetuado por um consumidor (provedor) para o fornecedor (consumidor), através de um *gateway* (terceira-parte); ou publicidade, onde o conteúdo da mídia é emitido por um fornecedor (provedor) para o consumidor (consumidor da publicidade); ou mesmo na entrega do produto, onde o produto é transportado do remetente (fornecedor) para o destinatário (consumidor).

Para cada uma das partes envolvidas (consumidor, provedor ou terceira-parte), o objeto serviço define a interface, assim como os repectivos comportamentos que os mesmos poderão apresentar. Assim, a Facilidade de Gerenciamento de Serviço inclui:

1. modelo de ciclo de vida de serviço: o ciclo de vida define o conjunto de estados em que um serviço pode se encontrar, assim como as possíveis transições em um determinado estado.

2. modelo de política: é o conjunto de políticas estabelecidas por todos os participantes que utilizam o serviço (e.g. nível de segurança).

3. modelo de participante: o participante é considerado como uma combinação dos atributos de identidade, domínio e função.

Facilidade de Contrato

Como estrutura de objetos, os contratos abrangem tanto o lado estático que expressa o conceito de contrato utilizado no mundo real, assim como o lado dinâmico, onde os contratos dão suporte à execução de processos relacionados aos Serviços Comerciais, regidos através de políticas mutáveis (dinâmicas).

De acordo com [9], a Facilidade de Contrato é uma extensão da especificação do serviço que contém os requisitos específicos dos contratos comerciais.

Facilidade de Apresentação de Objetos

A Facilidade de Apresentação de Objetos provê uma estrutura para apresentar e gerenciar componentes como: serviços, contratos e certificados, entre outros existentes.

A Facilidade de Navegação e de Apresentação de Objetos introduzem uma estrutura para que as entidades de comércio eletrônico possam ser examinadas, apresentadas e executadas, de forma a atender à requisição do consumidor.

O nosso modelo possui uma interface gráfica com o intuito de efetuar a comunicação com o consumidor que por sua vez descreve o produto desejado que será procurado no catálogo do vendedor.

2.4.3 Facilidade da Infra-estrutura de Mercado

Facilidade de Catálogo

O catálogo é um objeto estruturado que pode ser visualizado e transferido pela rede. A sua principal aplicação é conter informações de contratos e serviços. O catálogo provê uma interface que permite reordenar as suas entradas de diferentes formas, para que se possa incluir, modificar e excluir as informações existentes.

A diferença existente entre o objeto Catálogo e o serviço Catálogo é de que o objeto é algo que pode ser visualizado, consultado e transferido pela rede, enquanto que o serviço de Catálogo é o serviço que quando solicitado retorna o objeto Catálogo.

Os requisitos básicos do objeto Catálogo incluem:

- mecanismos de reordenação na exclusão e na inclusão de itens;
- uma interface para consulta;
- subsídios para que se tenha uma associação entre o catálogo e uma agência, e que seja explícita e não repudiável;

A infra-estrutura do catálogo deve suportar integração transparente e interoperável entre os catálogos provenientes de fontes diferentes. Diferentes implementações da Facilidade de Catálogo devem suportar a mesma interface.

A implementação dos catálogos em XML no nosso modelo tem como finalidade prover a interoperabilidade entre os diversos definidos por cada projetista de catálogo. Vale salientar porém que, a implementação em XML por si só não pode prover a interoperabilidade. Portanto, para garantir a interoperabilidade de catálogos em XML, diversos padrões são propostos (no Capítulo 3), sendo que um deles deve ser utilizado para que o catálogo seja interoperável entre diferentes fornecedores.

Facilidade de Corretagem

Os participantes no comércio eletrônico (consumidor, vendedor ou fornecedor), independente de sua função, lidam com informações e desempenham o papel de consumidor ou fornecedor de informação. Os consumidores de informações buscam e filtram informações, enquanto que os fornecedores enviam informações.

O principal propósito da Facilidade de Corretagem é permitir aos usuários (consumidores ou fornecedores de informações) uma certa facilidade no tratamento das informações, referentes aos serviços comerciais, inserido nos mercados eletrônicos globais. A presente Facilidade direciona os pedidos de informações nas fontes pertinentes.

A confidencialidade das informações é muitas vezes requerida pela grande maioria dos clientes e fornecedores que demandam anonimato nas transações comerciais. Um outro requisito dos participantes no comércio eletrônico bastante comum diz respeito à privacidade das informações pertencentes a cada participante.

Um mecanismo bastante utilizado para obter privacidade e anonimato é a implementação de uma terceira parte que garanta os requisitos demandados pelos participantes. Esta é uma das funções da Facilidade de Corretagem.

Facilidade de Agência

A Facilidade de Agência suporta os requisitos gerais para a padronização de um ponto comercial no mercado eletrônico. Na realidade, esta facilidade estabelece um ponto formal

de acesso e uma interface pública, que viabiliza o acesso e a consulta dos consumidores e fornecedores no mercado eletrônico.

A agência no nosso modelo é delimitada pela *região* definida na arquitetura de agentes móveis *Grasshopper* (Seção 2.3.2). Portanto, todos os componentes pertencentes a uma determinada *região*, são considerados como pertencentes a mesma agência.

Capítulo 3

O Uso de XML

3.1 Fundamentos

XML é um padrão proposto pela W3C (*World Wide Web Consortium*) utilizada para estruturar as informações na Web [13]. XML define a estrutura dessas informações através da tecnologia dos marcadores (*markups*), a mesma que é utilizada na linguagem HTML. Na realidade, tanto XML como HTML são derivados da mesma origem, a SGML (*Standard Generalized Markup Language*). Alguns autores afirmam que HTML (*Hiper Text Markup Language*) será substituído por XML, considerando-o como o HTML do futuro [21]. A grande diferença entre essas linguagens consiste na sua extensibilidade, ou seja, na flexibilidade quanto à criação de novos marcadores. Enquanto que XML possui marcadores extensíveis (flexíveis), onde novos marcadores podem ser criados e utilizados de acordo com a necessidade do usuário, HTML possui marcadores fixos, como <P>,<TABLE>, <BODY>, etc., de forma que não se pode alterar e muito menos criar novos marcadores, além do que já foi definido na própria linguagem [19].

O código a seguir apresenta as diferenças entre ambos os padrões (HTML e XML) que utilizam a tecnologia dos marcadores. Observe que no padrão XML novos marcadores ou tags foram criados para estruturar essas informações, dando semântica (significado) às mesmas.

```
<!--Exemplo em HTML-->

<h1>Catálogo de Automóveis</h1>

<p>Marca: FIAT
```

```
<p>Modelo: Palio EDX

<p>Ano:1999

<p>Cor: Azul Santiago Metálico

<p>Preco: R$ 16.000


<!--Exemplo em XML-->

<Automóvel>

<Marca>FIAT</Marca>

<Modelo nome = 'Palio' categoria = 'EDX' />

<Ano>1999<Ano>

<Cor nome = 'Azul Marinho' pintura = 'Metálica'/>

<Preço moeda = 'Real'>16.000</Preço>

</Automóvel>
```

Nota-se que novos marcadores foram criados no arquivo XML tais como Automóvel, Marca, Modelo, entre outros. Porém a utilização desses marcadores são regidos por regras e sintaxes que são validados através do arquivo DTD (*Document Type Definition*). Em outras palavras, o arquivo DTD define o formato e a sintaxe de cada marcador que foi criado no arquivo XML. Assim, cada arquivo XML possui o seu próprio arquivo DTD que irá definir cada marcador criado.

A seguir apresentaremos um exemplo de DTD para o arquivo XML codificado anteriormente.

```
(1) <!-- Exemplo de DTD para o arquivo XML ilustrado -->

(2) <!ELEMENT Automóvel (Marca,Modelo,Ano,Cor,Preço)>
```

```
(3) <!ELEMENT Marca (\#PCDATA)>

(4) <!ATTLIST Modelo nome (CDATA \#REQUIRED)>

(5) <!ATTLIST Modelo categoria (CDATA \#REQUIRED)>

(6) <!ELEMENT Ano (\#PCDATA)>

(7) <!ATTLIST Cor nome (CDATA \#REQUIRED)>

(8) <!ATTLIST Cor pintura (Normal|Metálica|Perolizada
''Metálica'' \#REQUIRED)>

(9) <!ATTLIST Preço moeda (CDATA \#REQUIRED)>
```

A Tabela 3.1 detalha as declarações do arquivo DTD acima.

Declaração	*Interpretação*
(2) <!ELEMENT Automóvel (Marca,Modelo,Ano,Cor,Preço)>	O marcador Automóvel contém cinco subelementos nessa seqüência
(3) <!ELEMENT Marca (#PCDATA)>	O elemento Marca é composto por uma cadeia de caracteres
(4) <!ATTLIST Modelo nome (CDATA #REQUIRED)>	O elemento Modelo possui um atributo nome com formato caracter, exceto "<", ">" e "&". Um valor deve ser fornecido
(5) <!ATTLIST Modelo categoria (CDATA #REQUIRED)>	O elemento Modelo possui um atributo categoria que é uma cadeia de caracteres. Um valor deve ser fornecido
(6) <!ELEMENT Ano (#PCDATA)>	O elemento Ano possui formato de uma cadeia de caracteres
(7) <!ATTLIST Cor nome (CDATA #REQUIRED)>	O elemento cor possui um atributo nome com o formato caracter e o atributo é requerido
(8) <!ATTLIST Cor pintura (Normal—Metálica—Perolizada) "Metálica"#REQUIRED)>	O elemento cor possui um atributo pintura cujo valor pode ser "Normal", "Metálica" ou "Perolizada". O valor "Metálica" sempre deve ser fornecido, caso nenhum valor tenha sido definido
(9) <!ATTLIST Preço moeda (CDATA #REQUIRED)>	O elemento Preço possui um atributo moeda que é requerido e possui formato de uma cadeia de caracteres

Tabela 3.1: Declaração e interpretação do arquivo DTD

3.2 Catálogos em XML

O uso de XML é particularmente interessante no projeto dos catálogos utilizados no comércio eletrônico, posto que o conteúdo dos mesmos são transportados para várias máquinas diferentes, obrigando-se a utilização de um padrão que seja interoperável entre

eles. Além disso, normalmente o conteúdo dos catálogos são consultados utilizando-se diversas formas de pesquisa. O uso de XML pode permitir que essas pesquisas se tornem mais eficazes na procura dos produtos pelos consumidores, uma vez que os marcadores no XML possuem semântica (significado) e não apenas um simples delimitador de palavras, como ocorre com o padrão HTML. Vale ainda salientar que a vantagem chave do XML no projeto dos catálogos para comércio eletrônico é a possibilidade de se transportar informações com significado, ao invés de apenas textos, como ocorre no HTML. A seguir iremos apresentar as principais vantagens do uso de XML no projeto dos catálogos [39, 38].

- **Simplicidade**

  A primeira vantagem do XML na definição dos catálogos diz respeito a sua simplicidade, particularmente se comparado com os catálogos que possuem formatos binários. Os catálogos em XML possuem formato baseado em caracter, assim como os formulários textos de uma tabela relacional. Tal simplicidade permite que qualquer editor leia e edite, além de permitir a visualização direta do seu conteúdo, sem a necessidade da conversão do seu formato original.

- **Flexibilidade**

  Os dados do catálogo em XML podem ser escritos uma vez e gerados em diversas mídias diferentes, como no caso o CD-ROM. Através do XSL (*eXtensible Style Language*), os conteúdos dos catálogos podem ser apresentados em formatos diferentes. Por exemplo, XSL pode converter os dados contidos no catálogo para o formato HTML, em forma de uma tabela, sem que haja a necessidade de se alterar o arquivo que contém os dados do catálogo. Portanto, a grande vantagem aqui do XML é a possibilidade de separar o conteúdo da apresentação.

- **Descrição específica das informações**

  O uso do XML permite aos projetistas especificar melhor as informações de cada produto que será armazenado no catálogo. Por exemplo, no XML é possível definir os marcadores preço, cor, marca, entre outros, que permitirão definir cada atributo especificamente, diferente do HTML onde os seus marcadores são fixos.

- **Consultas mais dinâmicas**

  Em virtude dos marcadores serem flexíveis, cada marcador pode ser definido para facilitar a consulta aos dados armazenados, permitindo que os dados do catálogo sejam ordenados por vários marcadores, provendo dessa forma uma pesquisa mais customizada ao consumidor.

- **Interoperabilidade**

Para muitos autores XML é a nova linguagem da Internet. Os últimos browsers já suportam esse padrão, abrindo um caminho para a interoperabilidade dos catálogos descritos nesse padrão. Além disso, por se tratar de uma linguagem que possui formato texto e não binário, o seu conteúdo torna-se legível em qualquer ambiente.

Apesar destas vantagens citadas, os catálogos XML por si só não suportam a interoperabilidade indispensável nas aplicações de comércio eletrônico. É necessário uma padronização quanto à definição dos marcadores para que diferentes corporações possam trocar as informações existentes em seus catálogos, assim como realizar transações entre elas (*ebusiness*). Além disso, é de suma importância que esses catálogos possuam os marcadores definidos com o mesmo nome para prover o mesmo significado nesses dados, pois assim os consumidores ou os próprios *e-brokers* (corretores eletrônicos) poderão mais facilmente consultar os produtos nos catálogos, para realizar comparações entre os valores obtidos. Um exemplo clássico seria a comparação de preços nos diversos catálogos, o que só poderia ser realizado mediante a padronização dos marcadores definidos em cada catálogo. Vale salientar ainda que, atualmente, os desenvolvedores de XML possuem o mesmo problema dos desenvolvedores de componentes: padronização do vocabulário utilizado para descrever as informações transmitidas entre aplicações diferentes [28].

Existem vários esforços com a finalidade de solucionar o problema exposto anteriormente. Esses trabalhos definem os padrões para suportar a interoperabilidade entre os catálogos XML. Descrevemos dois dos mais discutidos atualmente e que por isso merecem uma atenção especial.

RosettaNet: RosettaNet é um consórcio de empresas ligadas à tecnologia da informação (informática) com o objetivo de criar estruturas padrões especificamente para este nicho, de forma que essas empresas em parceria possam comprar e vender produtos de uma maneira mais fácil. Esta arquitetura propõe o uso de um dicionário mestre com as definições mais comuns das empresas, produtos e transações. O uso dos marcadores estabelecidos neste dicionário de acordo com uma estrutura pré definida permite o diálogo entre as empresas em parceria, chamado de Processo de Interface de Parceiros ou PIP. O objetivo desta arquitetura é estabelecer um número cada vez maior de termos comuns utilizados entre as empresas em parceria. O padrão para a troca de informações entre as empresas parceiras é o XML [33].

eCo Framework Project: Proposta pela CoomerceNet, que é um grupo formado por 35 empresas composto entre outras pela 3COM, IBM, American Express, HP, NEC, Microsoft e Intel. Este grupo de empresas de ramos diferentes apresenta uma estrutura de comércio eletrônico não proprietária e orientada a objetos para integrar os principais serviços existentes nos sistemas de comércio eletrônico. Seu principal objetivo é solucionar o problema da interoperabilidade entre os repositórios de

dados existentes nos sistemas de comércio, uma vez que esses repositórios utilizam os marcadores XML sem uma prévia padronização. Além disso, eCo trata da integração entre os diversos padrões de interoperabilidade emergentes atualmente como CBL da Commerce One, OTP (*Open Trading Protocol*), OFX (*Open Financial Exchange*), entre outros. A diferença desse padrão para o padrão RosettaNet é de que o padrão *eCo* provê a interoperabilidade para nichos diferentes, e não apenas para um. O padrão *eCo* consiste de dois componentes. O primeiro é a *Arquitetura eCo* que apresenta regras para que as transações possam suportar a interoperabilidade. Essa descrição é dividida entre os seguintes componentes: rede, mercados, negócios, serviços e interações (troca). Essa especificação detalha como cada um desses componentes deverá ser definido, de forma que a mesma possa prover a interoperabilidade. Outro componente diz respeito às *Recomendações semânticas eCo* que descreve os melhores mecanismos para desenvolver um catálogo baseado em XML. Essas recomendações definem um conjunto de regras para construção dos blocos de dados em XML, além de ditar normas quanto à definição dos documentos e marcadores XML, com a finalidade de facilitar a interoperabilidade entre os documentos já existentes [7].

Existem outros padrões que foram definidos com a finalidade de se estabelecer uma linguagem comum na semântica dos dados do catálogo. A diferença existente entre os dois padrões discutidos (*RosettaNet* e *eCo Framework*) é que o primeiro foi desenvolvido com a finalidade de padronizar as semânticas comumente utilizadas entre as empresas da área de tecnologia da informação. *ECo Framework* diferentemente de *RosettaNet* não possui um nicho proprietário. O padrão *eCo* trata da integração entre os diversos padrões existentes atualmente como o CBL, OTP e OFX.

O uso de XML na confecção de catálogos tem crescido. Durante a nossa pesquisa encontramos dois exemplos que já utilizam esse padrão:

- IntuiCat [20];
- Requisite Technology [31].

3.3 XML versus CORBA

A interoperabilidade é um dos requisitos para sistemas de comércio eletrônico, conforme observado na Seção 2.1.3. No nosso modelo, conforme será descrito nos próximos capítulos, os catálogos serão projetados em XML, com a finalidade de prover esta interoperabilidade. Nessa seção apresentamos algumas diferenças existentes entre XML e CORBA.

Em primeiros lugar, CORBA diz respeito à infraestrutura, enquanto que XML é um mecanismo para descrever a estrutura dos documentos. CORBA, conforme conceituado na Seção 2.3.3, é um *middleware*, composta por vários serviços, formando um ambiente distribuído, enquanto que XML é apenas uma linguagem que utiliza a tecnologia de marcadores extensíveis.

O uso de XML é adequado no gerenciamento de documentos, assim como em aplicações onde o conteúdo dos mesmos é exibido ao usuário. Em contrapartida, CORBA é adequada em aplicações onde o conteúdo do documento processado é apenas manipulado entre os programas, não havendo a necessidade de se exibir ao usuário. Isso se justifica em virtude de XML não possuir uma estrutura para a troca de mensagens entre os programas presentes no *middleware* CORBA, através dos seus serviços e facilidades.

O componente CORBA que mais se assemelha a XML é a linguagem de definição da interface (IDL), que possui a função de descrever as interfaces dos objetos em uma linguagem neutra, de forma que estas interfaces sejam legíveis a outras plataformas. XML, da mesma forma, pode ser utilizada para descrever as interfaces dos objetos em uma linguagem que hoje em dia é suportada pela maioria das plataformas, porém é válido ressaltar que XML, diferente de CORBA, não provê os serviços necessários para formar um ambiente distribuído. Na realidade, XML e CORBA são tecnologias complementares, ou seja, XML complementa as funcionalidades da CORBA.

3.4 Integração XML e Java

Vários artigos, como por exemplo [18], consideram Java e XML como par importante, em virtude dos mesmos serem componentes onde um complementa o outro. XML contribui provendo um formato de dados portável para qualquer plataforma, enquanto que Java provê uma linguagem que suporta a portabilidade de plataforma, posto que o seu código pode ser executado em qualquer ambiente, desde que possua a JVM (*Java Virtual Machine*) para aquele ambiente. A integração das duas tecnologias proverá uma plataforma com códigos e dados portáveis.

O objetivo de integrar a linguagem Java com o padrão XML é o desenvolvimento de aplicações para processar os documentos codificados em XML. Essa integração proverá um ambiente portável para qualquer plataforma e que permita compartilhar e processar dados.

A seguir citaremos algumas das vantagens da linguagem Java [21]:

Multiplataforma, a linguagem Java permite que o seu código seja executado em qualquer plataforma independente da arquitetura da CPU e do sistema operacional da máquina.

Alta produtividade, a tecnologia de orientação a objetos, assim como o uso dos *design patterns*, tornam esta linguagem mais produtiva.

Suporte nativo à Internet, a biblioteca de redes do Java contém rotinas bastante sofisticadas quanto à conexão com a Internet.

Suporte aos caracteres internacionais, assim como o XML, o Java utiliza o conjunto de caracteres da *Unicode* (caracteres de 2 bytes).

3.4.1 APIs de Integração

Existem duas APIs (*Application Programming Interface*) utilizadas para fornecer acesso às informações armazenadas nos documentos XML: SAX (*Simple API for XML*) e DOM (*Document Object Model*) [15, 16, 8]. Vale salientar que, apesar das vantagens da linguagem Java, as linguagens de programação C++, Perl, Pyton e Visual Basic também podem ser utilizadas para implementar um programa que manipule os documentos XML, utilizando as APIs SAX e DOM.

Tanto SAX como DOM foram criadas para a mesma finalidade, ou seja, prover acesso às informações existentes nos documentos XML, utilizando uma das linguagens mencionadas anteriormente. Porém é valido ressaltar que ambas possuem mecanismos diferentes de acesso às informações.

DOM (Document Object Model)

A API DOM provê acesso às informações armazenadas nos documentos XML como um modelo de objetos hierárquico, criando uma árvore de nós baseada na estrutura e na informação do documento XML. O acesso à informação é realizado através da interação com os nós da árvore criada. Em outras palavras, uma árvore (estrutura hierárquica) é criada para representar todos os elementos e atributos existentes em um documento XML, que será lido por um programa escrito em Java ou em uma outra linguagem qualquer.

Nos documentos, normalmente, a seqüência dos elementos é muito importante. Como DOM preserva essa seqüência existente nos arquivos XML, o mesmo é chamado como o modelo de objeto do documento. Vale ainda salientar o que DOM é a API recomendada oficialmente pela W3C, consórcio que projetou o XML.

A Figura 3.1 apresenta a árvore gerada a partir do nosso catálogo de automóveis definido na Seção 3.

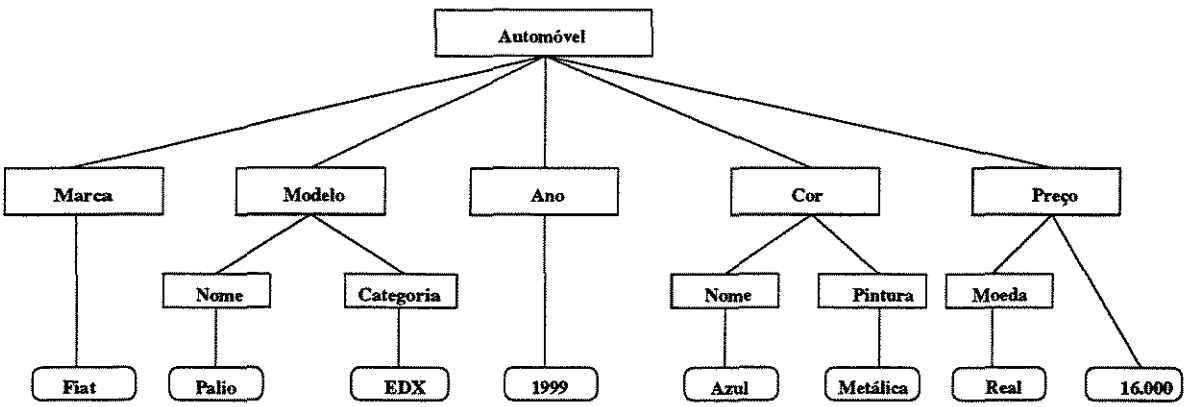


Figura 3.1: Estrutura hierárquica do catálogo de automóveis

SAX (Simple API for XML)

SAX é a API orientada a eventos que permite o acesso aos documentos em XML de forma mais simples e rápida. SAX é especialmente adequada para aplicações que realizam apenas consultas, sem que haja a necessidade de se alterar a sua estrutura. Normalmente o uso dessa API requer menos código e menos memória, posto que não há a necessidade de se alocar espaço para construir uma estrutura hierárquica (árvore) na mesma. O programa que implementa SAX apenas lê o arquivo XML e dispara alguns eventos baseados nas ocorrências que forem encontradas durante o decorrer da leitura no documento. Os eventos são disparados quando for encontrada uma das ocorrências relacionadas abaixo:

- Início e fim de um marcador;
- Início e fim do documento;
- Seções #PCDATA ou CDATA (seções onde o seu conteúdo é isolado do interpretador XML)
- Trecho contíguo de caracteres em um elemento.

SAX pode realmente ler o arquivo XML de forma mais rápida em virtude do mesmo não criar a estrutura hierárquica como ocorre no DOM. Por outro lado, o programador é obrigado a implementar o código que irá tratar cada evento disparado durante a leitura do documento.

Comparação das APIs

O uso do DOM é extremamente adequado quando se deseja modificar a estrutura ou mover partes do documento XML. Por exemplo, na Figura 3.2, temos um documento XML que será lido por um parser DOM que cria uma árvore correspondente ao documento lido. Após a criação dessa estrutura, o código lê os elementos e os atributos, adicionando e excluindo-os quando necessário, alterando dessa forma a estrutura inicial que o documento possuía.

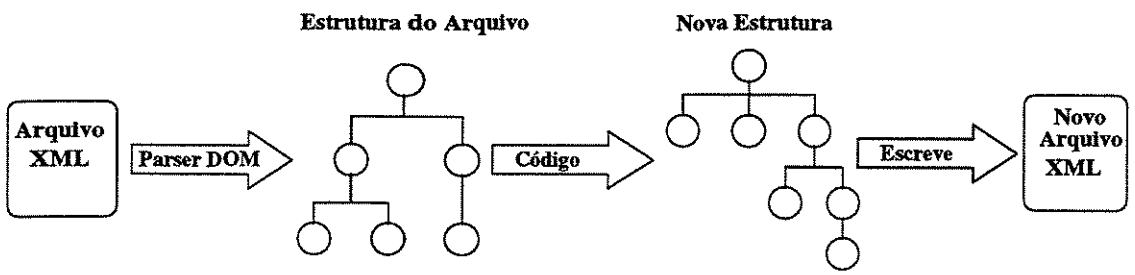


Figura 3.2: Processo de mudança da estrutura do Arquivo XML utilizando DOM

Como podemos observar, DOM é inadequado para processar um documento muito grande, dispondo de pouca memória. Se o arquivo a ser lido possuísse digamos 20 MB de tamanho, o uso do DOM seria inadequado, principalmente se formos consultar apenas alguns dados em um ambiente com pouca capacidade de memória.

SAX por outro lado, permite o acesso aos documentos XML de forma mais simples e mais rápida, o que leva a ter um melhor desempenho para realizar consultas nos documentos XML, principalmente se dispomos de pouca memória. Além disso, SAX, diferente do DOM que cria uma árvore hierárquica, proporciona uma melhor flexibilidade nos mecanismos de consulta, posto que, esta API permite a construção de seu próprio modelo de objetos que seja adequado ao contexto em que se aplica, utilizando-se para isso os tratadores de evento presentes nesta API. Vale ainda salientar que a aparente flexibilidade quanto à criação do próprio modelo de objeto pode se tornar um empecilho, uma vez que nessa API os desenvolvedores deverão se preocupar na construção do próprio modelo de objetos a ser utilizado.

3.5 Catálogos em XML e as Tecnologias de Comunicação Remota

A nossa pesquisa se baseia no uso dos catálogos XML para o ambiente de comércio eletrônico. Diversas tecnologias deverão ser utilizadas com a finalidade de proporcionar um ambiente de interoperabilidade, uma vez ser isso uma questão vital, quando o assunto é comércio eletrônico.

Várias ferramentas e tecnologias foram testadas com a finalidade de utilizar os melhores mecanismos para prover um ambiente de comércio eletrônico cujos catálogos estivessem escritos em XML. Apresentaremos nessa seção as ferramentas testadas para desenvolver um ambiente onde um consumidor possa procurar por produtos nos catálogos XML, assim como os resultados obtidos a partir dos testes.

3.5.1 RMI

O mecanismo de RMI apresenta algumas vantagens para o desenvolvimento do nosso trabalho. Para o desenvolvimento de catálogos em XML, acreditamos que o fato de podermos passar e receber os objetos como parâmetros poderá nos trazer vantagens no manuseio do catálogo no transporte dos dados entre máquinas remotas. Vale salientar também o melhor desempenho no tempo de resposta obtido por RMI. Quanto à implementação dos processos de negociação, RMI poderá trazer vantagens em função da sua portabilidade entre máquinas diferentes. A desvantagem desse mecanismo, está no fato de RMI estar limitado à linguagem Java.

3.5.2 CORBA

CORBA foi utilizada na nossa pesquisa em conjunto com os *applets*. Como os *applets* possuem restrições quanto ao uso dos recursos computacionais no servidor, o uso do mesmo para acessar os catálogos não foi possível. Solucionamos o problema através da utilização da CORBA no nosso ambiente de comércio eletrônico. Este uso é vantajoso tanto na definição dos catálogos como na implementação dos mecanismos de negociação automatizada, em virtude da mesma permitir o uso de diferentes linguagens em conjunto (interoperabilidade de linguagem). Isso obviamente só é possível em virtude do uso da linguagem comum IDL que define como cada objeto se comporta. Além disso, CORBA apresentou um dos melhores resultados quanto ao desempenho (próximo dos resultados obtidos por RMI). O uso da CORBA ainda pode prover os serviços disponíveis na mesma, como é o caso do Trader, que ampliará as funcionalidades do nosso ambiente de comércio eletrônico. Por outro lado, a complexidade da plataforma CORBA e o tempo necessário

para o aprendizado da mesma, assim como o aprendizado da linguagem IDL, podem ser considerados como desvantagens do uso desta plataforma.

3.5.3 Agentes Móveis Grasshopper

Os agentes móveis apresentaram alguns pontos positivos e negativos na nossa pesquisa, porém uma das vantagens que podemos ressaltar de imediato, diz respeito a sua mobilidade que não é limitada, como acontece com os mecanismos de chamada remota. Os agentes móveis permitem que um determinado código executável se mova entre várias máquinas, diferente do RMI, por exemplo que permite apenas chamar um método que está disponível em uma outra máquina. Ao contrário, os agentes podem se locomover para vários hosts antes de retornar à máquina origem. Essa mobilidade é muito útil nos sistemas de comércio eletrônico, uma vez que isso permite o desenvolvimento de um agente para percorrer e consultar vários catálogos, podendo inclusive negociar de forma automatizada com outros agentes que estejam a serviço dos fornecedores, retornando os resultados da pesquisa e negociação aos clientes que lançaram o agente de compra. A Figura 3.3 ilustra o percurso de um agente sendo transportado entre várias agências retornando os resultados à agência lançadora. É válido ressaltar neste ponto que o uso dos agentes móveis não trazem apenas vantagens, por exemplo, o tempo de mobilidade mais o tempo de leitura e processamento de um catálogo XML resultou no maior tempo se comparado com os outros mecanismos discutidos anteriormente.

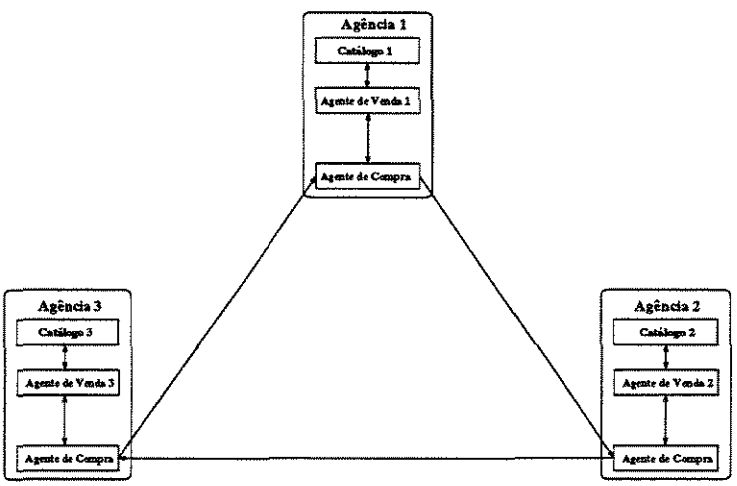


Figura 3.3: Mobilidade dos agentes entre máquinas diferentes

3.6 Resultados da Simulação

Esta Seção apresenta os resultados da simulação realizada com os catálogos em XML. Os programas da simulação foram codificados em *Java* utilizando o pacote *JDK* versão 1.2.2 (*Java Development Kit*). A integração Java e XML foi implementada através do uso das APIs DOM e SAX. O uso das duas APIs foi testada com a finalidade de verificar a sua performance quando integradas com cada uma das tecnologias de comunicação remota. Vale salientar que todas as simulações foram realizadas em uma estação *Sun Ultra SPARCStation 10* com 256 MB de memória RAM e sistema operacional *Solaris* versão 5.7.

A simulação diz respeito ao processamento remoto e local dos catálogos em XML. Para apresentação dos resultados desta parte, mostramos três gráficos que ilustram o tempo gasto para processar cada um dos catálogos de tamanhos diferentes.

Em seguida apresentamos as vantagens e as desvantagens de cada tecnologia de comunicação remota (RMI, CORBA e agentes móveis *Grasshopper*) na implementação dos aplicativos que acessam catálogos em XML. Posto que as aplicações de comércio eletrônico normalmente utilizam catálogos localizados em ambientes remotos, comparações dessas tecnologias com catálogos em XML seriam bastante interessantes no processo de desenvolvimento das aplicações de e-commerce.

3.6.1 Processamento Remoto de Catálogos em XML

Esta parte da simulação focaliza no processamento remoto das informações contidas nos catálogos em XML. Para esta finalidade vários programas foram implementados utilizando cada uma das tecnologias e as APIs de integração Java e XML (DOM e SAX). Além disso, implementamos cinco catálogos de tamanhos diferentes (53, 833, 1665, 3329 e 6657 elementos) para simularmos o seu comportamento diante do aumento das informações contidas no catálogo. A Figura 3.4 ilustra o tempo decorrido utilizando cada uma das tecnologias remotas para acessar e ler (através da API DOM) o catálogo inteiro localizado remotamente. A Figura 3.5 apresenta o gráfico contendo os tempos para realizar a mesma leitura no catálogo utilizando a API SAX.

Vale ressaltar que o tempo nos gráficos da Figura 3.4 e 3.5 corresponde à média de tempo obtida em 18 simulações executadas. Essa média foi calculada em virtude dos tempos de simulação apresentarem valores diferentes decorrentes principalmente do tráfego na rede no momento da simulação. Vale ressaltar que as simulações foram realizadas em horários diferentes, durante o período com um menor tráfego de rede, no intuito de minimizar o máximo possível estas diferenças resultantes. Dentre os que tiveram maiores diferenças, podemos citar a simulação com os agentes móveis que resultou em diferenças significativas, principalmente no processamento de catálogos menores. A Tabela 3.2 apre-

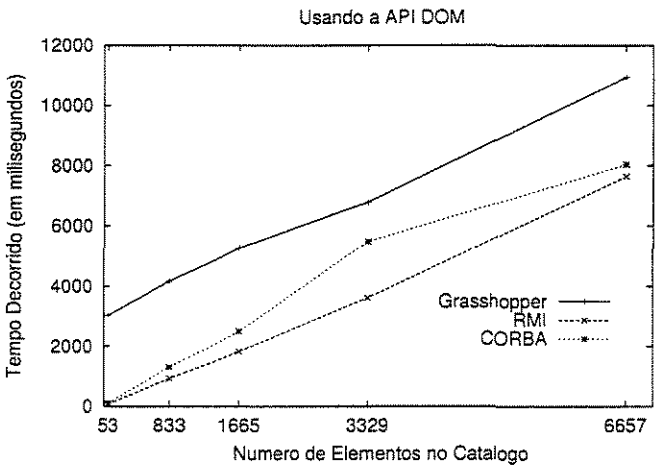


Figura 3.4: Utilização da API DOM para Catálogos Remotos

n	$\overline{x}$	δ
53	3.027	45,2
833	4.163	352,7
1.665	5.271	393,8
3.329	6.790	585,1
6.657	10.940	575,4

Tabela 3.2: Média e Desvio Padrão da simulação de *Grasshopper* e DOM

senta a média em milisegundos ($\overline{x}$) e o desvio padrão (δ) dos tempos obtidos na simulação executada, utilizando-se os agentes móveis *Grasshopper* e a API DOM. Em contrapartida, as Tabelas 3.3(a) e 3.3(b) apresentam respectivamente, os valores obtidos na simulação de RMI (Tabela 3.3(a)) e de Corba (Tabela 3.3(b)), utilizando-se a mesma API DOM. Nas Tabelas ilustradas, n diz respeito ao tamanho do catálogo, cujo valor representa a quantidade de elementos no catálogo. As Tabelas 3.4 e 3.5(a) e 3.5(b) apresentam os valores da simulação ilustrada no Gráfico 3.5 que utiliza a API SAX juntamente com os agentes móveis *Grasshopper* (Tabela 3.4), com RMI (Tabela 3.5(a)) e com Corba (Tabela 3.5(b)).

Conforme pode ser visualizado nas Figuras 3.4 e 3.5, SAX proporcionou melhor desempenho na leitura dos catálogos, como pode ser atestado pelo seu menor tempo. Além disso, SAX requereu menos código do que DOM para codificar o mesmo programa.

n	$\overline{x}$	δ
53	70	16,8
833	922	32
1.665	1.835	83,9
3.329	3.617	188,6
6.657	7.638	581

n	$\overline{x}$	δ
53	105	48,2
833	1.303	81,5
1.665	2.507	213,4
3.329	5.475	361,1
6.657	8.038	825,9

Tabela 3.3: Média e Desvio Padrão simulando DOM e (a) RMI, e (b) Corba

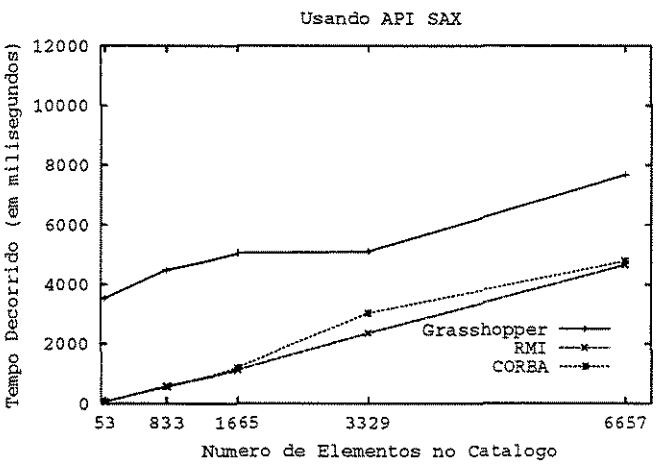


Figura 3.5: Utilização da API SAX para Catálogos Remotos

RMI

No mecanismo do RMI, foi implementado um programa cliente e um método remoto no servidor para ler as informações contidas no catálogo. Na nossa simulação o programa cliente invoca o método remoto localizado no servidor para ler e processar as informações contidas no catálogo. Para simularmos o comportamento de ambas as APIs, foram desenvolvidos dois métodos remotos diferentes no servidor, de forma a analisar separadamente o comportamento de ambas as APIs, assim como o desempenho do próprio RMI na leitura e processamento dos catálogos em XML.

CORBA

A implementação de CORBA foi semelhante à implementação de RMI, exceto que no caso da CORBA foi criada uma interface adicional (IDL) utilizada para descrever o objeto

n	$\overline{x}$	δ
53	3.532	505,2
833	4.485	498,6
1.665	5.046	65,9
3.329	5.104	258,7
6.657	7.681	501,4

Tabela 3.4: Média e Desvio Padrão da simulação de *Grasshopper* e SAX

n	$\overline{x}$	δ
53	54	7,4
833	584	74,4
1.665	1.123	75,7
3.329	2.357	156,2
6.657	4.655	253,1

n	$\overline{x}$	δ
53	58	21
833	560	73,7
1.665	1.200	102,2
3.329	3.030	816,7
6.657	4.786	218,9

Tabela 3.5: Média e Desvio Padrão simulando SAX e (a) RMI, e (b) Corba

remoto. Na CORBA também foram codificados dois métodos diferentes (um utilizando SAX e o outro DOM) com o objetivo de verificar o processamento utilizando ambas as APIs. Assim como no RMI, um programa cliente foi codificado para realizar a chamada e o processamento das informações contidas no catálogo.

Agentes Móveis Grasshopper

Para simularmos os agentes móveis na leitura das informações contidas no catálogo XML, foi implementado apenas um programa agente capaz de se locomover de um *host* a outro. Para rodar a nossa simulação com o agente móvel, inicializamos duas *agências* diferentes com a finalidade de suportar a sua execução tanto no endereço de origem como no de destino. Na nossa simulação o agente migra para uma máquina diferente para ler o catálogo e retorna para a máquina de origem. Foram implementados dois agentes diferentes com a finalidade de simular o uso de ambas as APIs (DOM e SAX).

3.6.2 Processamento Local de Catálogos em XML

Na simulação local foram implementados dois programas (um utilizando SAX e outro DOM) para acessar e ler o catálogo inteiro localmente. Conforme pode ser visualizado no gráfico da Figura 3.6, SAX proporciona uma leitura mais rápida para catálogos maiores,

enquanto que DOM proporciona leituras mais rápidas para catálogos menores. No código implementado para a simulação local, SAX também requereu menos código do que DOM para ler os mesmos catálogos.

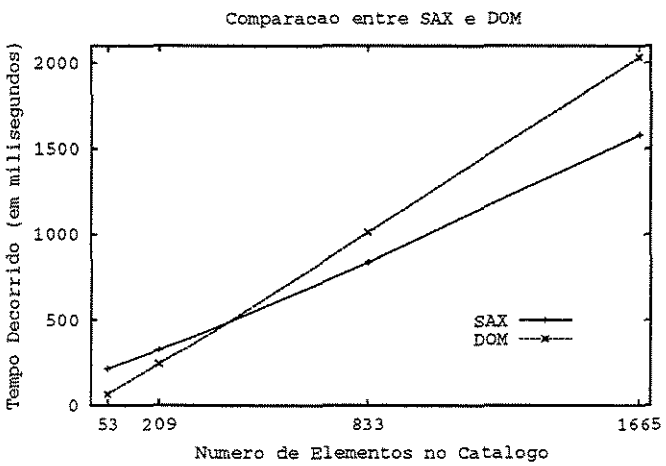


Figura 3.6: Processamento Local usando API SAX e DOM

3.7 Comparação das Tecnologias de Comunicação Remota

Analisando os resultados da simulação podemos afirmar que CORBA é a melhor opção no que diz respeito ao acesso aos catálogos remotos. CORBA alcançou um tempo de resposta bem próximo dos resultados de RMI para ler os catálogos em XML. Porém é válido ressaltar que CORBA possui serviços e facilidades bastante úteis nas aplicações de comércio eletrônico como é o caso do *Trader*, considerado no nosso protocolo de negociação. Além disso, CORBA permite que a interface dos objetos seja descrita em IDL, provendo dessa forma a interoperabilidade dos objetos entre linguagens diferentes, o que nas aplicações de comércio eletrônico é um requisito indispensável conforme já foi descrito na Seção 2.1.3.

Em contrapartida, o agente móvel provê uma codificação mais simples na confecção dos programas que tratam da mobilidade de objetos. Isso ocorre em virtude da grande maioria das plataformas possuírem intefaces e métodos implementados, bastando que as aplicações dos usuários utilizem os respectivos métodos para migrarem de uma máquina a outra. Muitas das plataformas, como é o caso do *Grasshopper*, permite que o objeto se

mova de um *host* a outro, bastando que para isso informe o endereço de destino do objeto a ser migrado.

Os agentes móveis, diferentes do RMI e CORBA, permitem que os objetos migrem de um host a outro de forma mais simples, uma vez que o processo migratório não requer o retorno ao endereço de origem antes de migrar a um outro endereço. Essa característica permite que um agente migre para diversos *hosts* fazendo uma pesquisa de preço de um determinado produto até que se alcance um preço que seja condizente com a requisição do consumidor. É válido ressaltar ainda que os agentes móveis são mais adequados em aplicações que envolvam negociação, uma vez que em função da exaustiva quantidade de propostas e contrapropostas ofertadas pelas aplicações, o tráfego da rede tende a aumentar, sendo assim, a migração para o *host* do fornecedor diminuiria o tráfego na rede para essas aplicações. Além disso, acreditamos que o uso de agentes móveis é mais adequado para modelarmos o comportamento do consumidor que se move para algum lugar para negociar e comprar algum produto. Com o uso dos agentes móveis é possível também que os agentes de compra se comuniquem entre eles de forma a informar a melhor oferta encontrada.

A Tabela 3.6 apresenta as vantagens e as desvantagens de cada uma das tecnologias abordadas nessa Seção.

Tecnologia	Vantagens	Desvantagens
RMI	Referência a objetos remotos; independência de plataforma proporcionada por Java; melhor tempo de resposta	Limitado à linguagem Java; mobilidade limitada
CORBA	Interoperabilidade de linguagem suportada pela IDL; serviços e facilidades disponíveis	Complexidade da IDL e da própria CORBA
Agentes Móveis Grasshopper	Mobilidade; paralelismo na leitura de diversos catálogos, além da diminuição do tráfego em função da mobilidade	Tempo de latência na mobilidade

Tabela 3.6: Comparação de RMI, CORBA e agentes móveis *Grasshopper* na leitura de Catálogos em XML

Capítulo 4

Negociação em Comércio Eletrônico

4.1 Introdução

As atividades de comércio na Internet têm-se limitado apenas na venda baseada em catálogos. Porém este cenário irá rapidamente se alterar no decorrer do tempo com a inclusão dos mecanismos de negociação para determinar os preços dos produtos e de outros atributos que sejam negociáveis. A negociação no comércio eletrônico pode resultar em grandes volumes de negócio entre clientes em potencial em um curto período de tempo com um custo inferior aos outros mecanismos de negociação convencionais utilizados hoje em dia. Além disso, a negociação no comércio eletrônico provê uma maior flexibilidade aos consumidores e fornecedores, em virtude de alguns atributos não serem pré-fixados.

Existem vários mecanismos de negociação, que vai desde os leilões eletrônicos (negociação entre vários participantes), até os mecanismos de negociação bilateral (negociação entre dois participantes). Existem também os mecanismos de negociação automatizada, onde as funcionalidades de negociação são efetuadas com a mínima intermediação humana possível, utilizando-se os recursos e as informações disponíveis. A negociação automatizada pode ser dividida em duas abordagens [3]:

- Negociação com aprendizado
- Negociação sem aprendizado

4.2 Negociação Automatizada

4.2.1 Introdução

A negociação automatizada no comércio eletrônico é definida como o processo, onde dois ou mais participantes barganham entre eles, utilizando as ferramentas das aplicações de

comércio eletrônico (automatizada), ao invés de se ter a negociação pessoa-a-pessoa. Essa modalidade de negociação deverá suportar os mecanismos automatizados para prover a proposta e a contraproposta [4].

A negociação automatizada vai desde a confecção dos agentes até a programação desses agentes com as estratégias de negociação, fornecendo aos mesmos as informações necessárias, permitindo assim uma certa autonomia para que esses agentes possam negociar com um indivíduo, ou mesmo com outros agentes, finalizando com um acordo ou a rejeição das propostas, o qual pode ou não requerer a aprovação final de uma pessoa que seja responsável pela sessão da negociação [4].

A seguir apresentaremos o uso dos agentes inteligentes para automatizar o processo de negociação. Como foi citado anteriormente existem duas abordagens no uso desses agentes: os agentes com aprendizado e os agentes sem aprendizado.

4.2.2 Agentes Inteligentes

O uso dos agentes inteligentes para automatizar o processo de negociação irá envolver a criação de um ou mais agentes, cada um deles possuindo um repertório de tarefas a serem executadas, que irão negociar de forma eletrônica no ambiente governado pelas regras definidas [4].

Algumas abordagens de negociação focalizam os seus estudos mais nos próprios agentes, neste caso, os agentes aprendem as estratégias de negociação, como acontece com o uso dos algorítmos genéticos utilizados para ensinar o processo de negociação aos agentes. Outras abordagens focalizam os seus estudos mais nas regras definidas, exigindo pouco processamento dos agentes. Nesta abordagem, normalmente, existe uma interface humana que determina as regras e os mecanismos de negociação, assim como as melhores estratégias a serem seguidas [3].

Existem duas linhas de pesquisa que determinam como os agentes devem adquirir e executar as instruções relacionadas à negociação. A primeira linha afirma que os agentes devem ser criados contendo neles um conjunto de estratégias já definidas. Neste caso, o agente deverá possuir uma boa capacidade de memória, que seja suficiente para armazenar todas as instruções necessárias para tratar cada situação encontrada no decorrer da negociação. A segunda linha afirma que os agentes devem ser capazes de aprender, ao invés de armazenar as regras. Esta linha de pesquisa afirma que os agentes devem ser capazes de adquirir experiências baseadas nas negociações anteriores [3, 2]. A seguir apresentamos modelos relacionados às duas abordagens.

Agentes Inteligentes sem Aprendizado

No artigo de Sandholm [34], o autor apresenta uma extensão do protocolo Contract Net que foi denominada de TracoNet (*Transportation Cooperation Net*). TracoNet é um modelo de negociação que é baseado no cálculo do custo marginal (otimização). A negociação neste modelo se processa no ciclo anúncio-oferta-concessão, onde um agente *anuncia* o frete desejado, informando entre outros dados, o destino da entrega. Após este anúncio, cada centro de despacho, situado em local diferente, *oferta* o valor do frete, que é calculado a partir do seu custo marginal para efetuar a entrega. O agente gerenciador recebe cada oferta concedida em cada centro de despacho e *concede* o frete para o centro que propôs a melhor oferta.

No artigo [6], os autores apresentam o *Kasbah*, um protocolo desenvolvido para negociar a compra e a venda dos produtos, utilizando os agentes inteligentes. Nesse modelo, segundo os autores, os agentes não são muito "inteligentes", como também não utilizam os mecanismos de aprendizado ou as técnicas da inteligência artificial. Ao invés disso, no *Kasbah*, os agentes recebem todas as estratégias de negociação dos usuários, através dos formulários preenchidos pelos consumidores que especificam dentre os inúmeros detalhes, o intervalo de preço aceitável do produto. Neste artigo os autores demonstram os resultados práticos baseados nas experiências realizados com alguns usuários.

Agentes Inteligentes com Aprendizado

Zeng e Sycara apresentam Bazaar [41], um sistema experimental para atualizar as ofertas em uma sessão de negociação bilateral envolvendo agentes inteligentes. Este artigo apresenta uma modelagem de negociação, utilizando a probabilidade *Bayesiana*, como mecanismo de aprendizado. Os autores apresentam um exemplo de negociação de preço, no referido modelo.

O estudo dos algorítmos genéticos tem se mostrado como bastante promissor para ser aplicado na área da negociação automatizada. Os algorítmos genéticos são baseados na evolução Darwiniana, e no contexto da negociação funcionam da seguinte forma: inicialmente, cada um dos agentes gera de forma randômica outros agentes com as estratégias de negociação que não necessitam ser as melhores. A cada geração dos agentes, a performance de cada estratégia de negociação é verificada, de forma que se gerem filhos dos agentes possuindo estratégias melhores. A principal desvantagem do uso dos algorítmos genéticos é o número de testes necessários para se obter no final uma boa estratégia de negociação. Esse número pode variar entre 20 gerações [23] até mais de 4.000 gerações [2].

Oliver em [23] apresenta a aplicação dos algorítmos genéticos no ensino da negociação aos agentes. Oliver mostra neste mesmo artigo o aprendizado das melhores estratégias de negociação, após repetidos treinamentos, inclusive para negociações consideradas como

complexas.

4.3 Modelo de Negociação do OMG

O modelo de negociação do OMG para comércio eletrônico baseado em CORBA apresenta três tipos de negociações: negociação bilateral (negociação entre dois participantes), multilateral (negociação entre vários participantes) e a promissória (modelo proposto para negociar alguma dívida existente entre o consumidor e o fornecedor) [26]. A seguir descreveremos cada um desses modelos.

4.3.1 Negociação Bilateral

A negociação bilateral é um modelo projetado para que dois participantes em uma sessão de negociação possam interagir com a finalidade de alcançar um acordo mútuo. O modelo possui três estados intermediários: *requested* (requisitado), *proposed* (proposto) e *offered* (ofertado), os quais, através da sua interação, pode levar a um dos estados terminais: *agreed* (acordo), *rejected* (rejeitado) ou *timeout* (estouro de tempo). A Figura 4.1 ilustra o funcionamento do modelo.

A negociação bilateral pode ser iniciada em um dos três estados: *proposed*, *requested* ou *offered*. *Offered* indica um estado em que a proposta da negociação não pode ser alterado, enquanto que *propose* lança uma proposta, permitindo a alteração da proposta de negociação, quando necessário. Na verdade, ambos os estados apresentam grandes possibilidades para que se chegue a um acordo mútuo entre os participantes. Por outro lado, o estado *requested* denota um estado onde não há nenhum compromisso entre os participantes. A transição *suggest* é utilizada para invocar sugestões um ao outro. Neste modelo é fixado um tempo pré-determinado em que o sistema pode ficar inativo, porém caso este tempo limite de inatividade venha a ser ultrapassado, a transição *timeout* será invocada.

Transições

Request

Request permite a alteração da proposta de negociação assim como a alteração do estado *proposed* para o *requested*. A transição *request* não implica em um compromisso entre os participantes da negociação, porém a mesma dá abertura para que a outra parte responda com *propose* ou *offer*.

Suggest

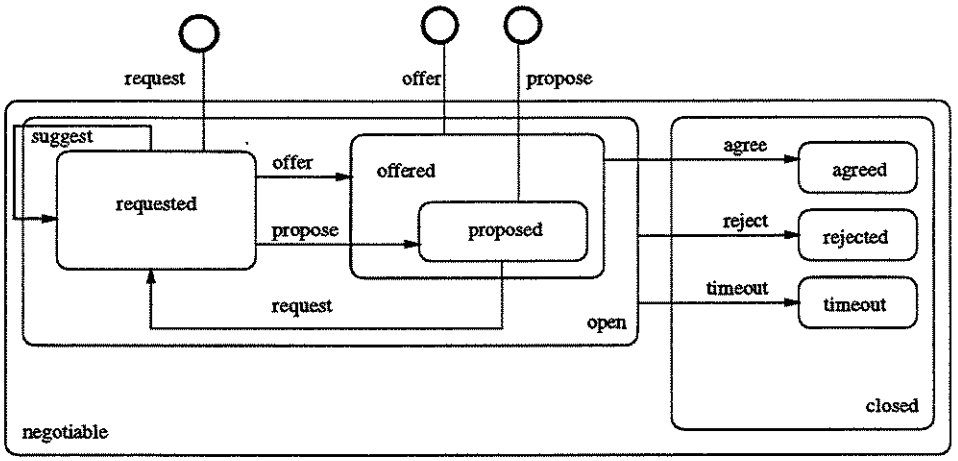


Figura 4.1: Modelo Bilateral

Suggest é semanticamente semelhante à transição *request*, exceto que a transição *suggest* é iniciada no estado *requested*. Esta transição é utilizada para solicitar novas propostas, uma vez que o tempo de inatividade pode levar ao estado final *timeout*. Nesta transição uma das partes envolvidas invoca por sugestões a outra, até que uma delas mude para um nível de compromisso maior definido através dos estados *proposed* e *offer*.

Propose

Propose é uma transição do estado *requested* para o *proposed*. Esta transição é chamada quando uma das partes envolvidas na negociação deseja ofertar uma proposta. Vale ressaltar que esta transição altera o estado da negociação para um estado com um compromisso maior entre as partes envolvidas.

Offer

Offer é uma transição do estado *requested* para o *offered*. A transição *offer* é bastante semelhante à transição *propose*, uma vez que ambas as transições são invocadas para ofertar uma proposta. A diferença básica existente entre elas é que *offer*, diferente de *propose*, não permite que a proposta seja alterada. Em outras palavras, na transição *propose* a proposta poderá ter futuras alterações e negociações através das transições *propose* e *request*, enquanto que *offer* não permite futuras negocições quanto à proposta ofertada.

Agree

Esta transição disponível no estado *offered* é invocada para selar o acordo entre ambos os

participantes da negociação.

Reject

A transição *reject* é chamada a partir do estado *open*, sendo assim, esta transição poderá ser invocada de qualquer sub-estado de *open* (*offered*, *proposed* ou *requested*). *Reject* é chamada para abortar a negociação com a outra parte ou quando se deseja negar a proposta ofertada.

Timeout

Timeout é uma transição que poderá ser invocada em qualquer sub-estado do estado *open* e a mesma é chamada quando o tempo de inatividade ultrapassar o tempo máximo pré-estabelecido nessa sessão.

Estados

Open

É o estado em que a negocição se encontra em aberto. Esse estado contém três sub-estados: *offered*, *proposed* e *requested*. A transição *reject* assim como a *timeout* são válidas neste estado.

Offered

Nesse estado o participante da negociação pode aceitar (através da transição *agree*) ou rejeitar (*reject*) a proposta que foi ofertada.

Proposed

Proposed estende semanticamente o estado *offered*, introduzindo a possibilidade de negociar a proposta ofertada. A negociação e a alteração da proposta podem ser realizadas através da transição *request* que permite retornar ao estado *requested* onde se encontra um compromisso menor entre os participantes da negociação.

Requested

O estado *requested* é o estado onde encontramos o menor grau de compromisso entre os participantes da negociação. Semanticamente, este estado sugere novas propostas para que as negociações entre ambas as partes possam prosseguir.

Agreed

Agreed é um estado terminal de sucesso, indicando que a proposta ofertada foi aceita pelos participantes da negociação.

Rejected

Rejected é um estado terminal de falha que indica que a proposta ofertada foi rejeitada por um dos participantes da negociação, ou que a sessão da negociação foi abortada.

Timeout

Timeout é um estado terminal de falha indicando que a sessão da negociação foi finalizada, em virtude da inatividade durante um determinado período de tempo pré-estabelecido.

Exemplo

A seguir ilustramos um exemplo de funcionamento do modelo bilateral de forma a esclarecer cada estado e transição.

O consumidor requisita (através da transição *request*) uma proposta para a compra de um produto, informando as características desejadas, indo assim para o estado *requested*. A partir deste estado, existem duas possibilidades de transição:

- O fornecedor pode lançar uma oferta final através da transição *offer*, migrando-se assim para o estado *offered* cuja oferta pode ser aceita (utilizando a transição *agree*), finalizando a negociação no estado final *agreed*. A oferta pode ser rejeitada através da transição *reject* que por sua vez vai para o estado final *rejected*.

- O fornecedor pode lançar uma oferta negociável através da transição *propose* migrando para o estado *proposed*. Em virtude da oferta ser negociável, o consumidor pode neste estado (*proposed*) requerer através da transição *request* mudanças na oferta proposta, retornando ao estado anterior *requested*. A partir deste estado (*requested*) ambos os participantes da negociação (consumidor e fornecedor) podem requerer por novas ofertas através da transição *suggest*, assim como ofertar novas propostas.

Vale salientar duas observações importantes sobre o modelo:

1. A transição *timeout* pode ser disparada a partir de qualquer estado. Assim, caso os participantes da negociação estejam inativos por um determinado período de tempo, a transição *timeout* será invocada;

2. O consumidor ou o fornecedor pode rejeitar, através da transição *reject*, o prosseguimento da negociação no decorrer do tempo em que a sessão da negociação estiver aberta, diferente da transição *agree* que só pode ser invocada a partir do estado *offered*.

4.3.2 Negociação Multilateral

A negociação multilateral, ilustrada na Figura 4.2, é um modelo que permite a negociação entre dois ou mais participantes. Este modelo provê uma estrutura que permite o lançamento de uma proposta, obtendo-se o acordo após a sessão de negociação, através de um consenso entre os participantes na proposta ofertada.

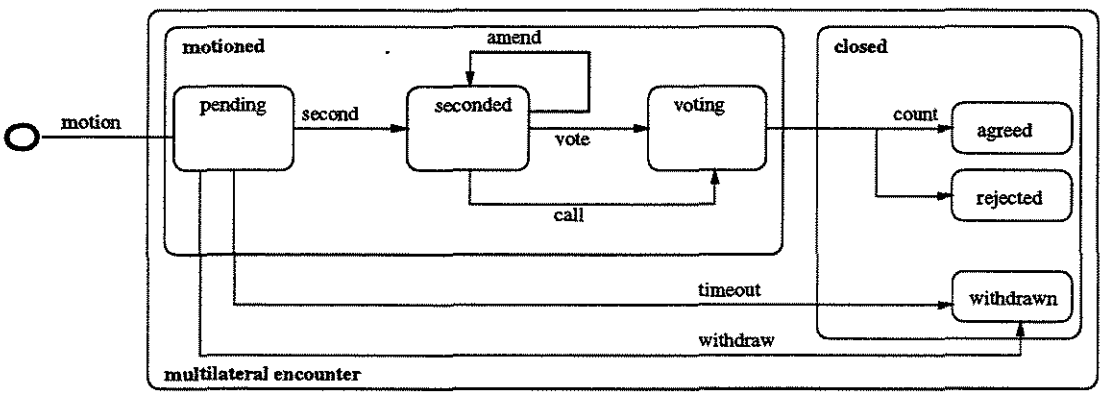


Figura 4.2: Modelo Multilateral

O modelo é composto por três estados principais denominados de *pending*, *seconded* e *voting*, que através das interações entre os participantes podem levar a um dos estados terminais *agreed*, *rejected* ou *withdrawn*. A inicialização da negociação neste modelo é efetuada invocando-se a transição *motion* que indica uma proposta ofertada por um dos participantes.

No estado *pending*, três ações podem acontecer:

- o participante pode retirar a proposta por ele ofertada;
- qualquer participante que não seja o próprio autor pode apoiar a proposta ofertada;
- a proposta pode ser retirada através da transição *timeout*, por falta de apoio.

As propostas ofertadas e apoiadas por algum outro participante que não seja o próprio autor passa do estado *pending* para o *seconded*. Vale ressaltar que a partir desse estado uma contagem regressiva para a votação é ativada com a finalidade de calcular o número de apoios recebidos na proposta por outros participantes da negociação. Durante o estado *seconded*, qualquer participante pode chamar a transição *amend* ou a *call* antes que a transição *vote* seja invocada. *Amend* é utilizado para efetuar alterações nas ofertas lançadas. Uma vez tratar de uma negociação multilateral, qualquer alteração nas propostas só será processado (através da transição *amend*) se houver o consenso dos participantes da sessão.

Call é chamada para forçar a votação da proposta em negociação, enquanto que a transição *vote* é invocada regida por um temporizador que migra para o estado *voting* após um determinado período de tempo. No estado *voting*, cada participante da negociação é obrigado a votar respondendo: sim, não ou abstenção. Após a votação cada voto é conferido com a finalidade de aceitar ou rejeitar definitivamente a proposta ofertada, alcançando os estados terminais *agreed* ou *rejected* de acordo com o resultado da votação.

4.3.3 Modelo de Promissória

O modelo de Promissória define a negociação no cumprimento de alguma promissória (dívida) existente entre um devedor e um credor. A Figura 4.3 ilustra o modelo de promissória que define uma seqüência de interações para que o devedor da promissória cumpra o compromisso estabelecido com o credor.

O processo nesse modelo é inicializado quando um indivíduo chama a transição *promise* compromentendo-se com alguma promissória. Esta transição (*promise*) muda o estado para *right*. A transição *expire* é chamada quando o credor deixa de requerer o seu direito junto ao seu devedor por um determinado período de tempo, expirando o seu direito, levando ao estado *expired*. Uma vez inicializado, o credor pode requerer o seu direito junto ao devedor através da transição *request*, o que torna a promissória como pendente, alterando o seu estado para *pending*. O modelo também pode ser inicializado através da transição *commit*. Quando incializada aravés da transição *commit*, a promissória é automaticamente considerada como pendente, migrando o seu estado para o *pending*. Caso o devedor não cumpra a promissória por um determinado período de tempo, esta promissória passará para o estado *overdue* que semanticamente indica que a promissória está vencida.

O devedor da promissória cumpre o compromisso através da transição *fulfill* que na realidade é uma transição que invoca os mecanismos da negociação *bilateral* ou *multilateral* no cumprimento da promissória. Sucesso na negociação leva ao estado *fulfilled*, enquanto que a falha na negociação leva ao estado *rejected*. A transição *waive* serve para abdicar dos direitos adquiridos anteriormente. O processo de abdicação envolve os mecanismos de negociação que são baseados no modelo *bilateral* ou *multilateral*.

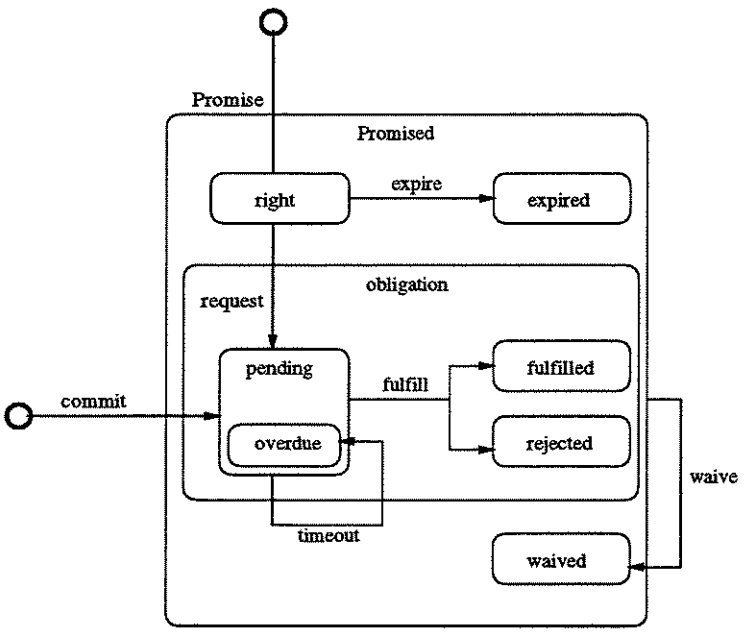


Figura 4.3: Modelo de Promissória

Capítulo 5

Modelo Proposto para Negociação Automatizada

5.1 Visão Geral

Este capítulo apresenta o modelo de negociação proposto, detalhando as funcionalidades, assim como a sua estrutura. A nossa abordagem de similaridades para determinar as semelhanças entre os produtos ofertados também é discutida neste capítulo.

No nosso modelo, o comprador e o vendedor são representados através dos agentes móveis para permitir a mobilidade dos mesmos. O agente de compra migra para a máquina do agente de venda, minimizando o uso da rede na comunicação entre eles. Na negociação, a mobilidade é bastante interessante, posto que na grande maioria das vezes, este processo envolve uma exaustiva permuta de ofertas e contra-ofertas.

O protocolo de negociação é composto de cinco fases: *chamada, seleção, negociação, apresentação* e *conclusão.* A negociação tem início quando o agente de compra faz a *chamada* por propostas, enviando os dados do produto requisitado pelo consumidor para o agente de venda. A *seleção* diz respeito à seleção dos agentes de venda que retornaram propostas potencialmente aceitáveis ao agente de compra. A *negociação* compreende a barganha entre o agente de compra e os de venda que foram previamente selecionados. Após esta fase, os resultados gerados através da negociação são *apresentados* ao consumidor que por sua vez rejeita ou aceita a oferta proposta, *concluindo* a sessão de negociação. Vale salientar que a sessão de negociação pode se repetir pelo número de vezes que o consumidor desejar.

O protocolo proposto é composto pelo agente de compra, agentes de venda e pelos catálogos localizados nos *hosts* dos respectivos agentes de venda. Conforme descrito na Seção 3.7, o uso dos agentes móveis trazem algumas vantagens interessantes para o nosso modelo. Citaremos a seguir algumas dessas vantagens que valem a pena serem ressaltados:

- Economia no tráfego da rede. Em virtude da grande quantidade de propostas e contrapropostas emitidas pelo nosso modelo de negociação, a mobilidade para o *host* onde será realizado a negociação se torna vantajosa para economizar o tráfego na rede.
- Facilidade na implementação de programas que migrem de uma máquina a outra. A maioria das plataformas já possuem classes e métodos que auxiliam em processos migratórios.
- Diferente de RMI e CORBA, os agentes móveis permitem com uma maior facilidade que os objetos migrem de um *host* ao outro sem a necessidade de retornar ao *host* original.

As seções do presente capítulo abordam os assuntos a seguir:

Diagrama de Transição de Estados: Apresenta o Diagrama de Transição de Estados do modelo de negociação automatizada proposto, indicando as principais diferenças existentes em relação ao modelo de negociação Bilateral proposto pelo OMG, abordado na Seção 4.3.1.

Medidas de Similaridades: O mecanismo utilizado no nosso modelo para determinar os produtos semelhantes será abordado neste Capítulo. Este mecanismo é de grande importância para o nosso modelo, uma vez que ela determina os produtos semelhantes de forma que se possa ofertar um produto semelhante ao requisitado pelo consumidor, caso o produto com as características exatas não possa ser encontrado.

Agentes de Compra e Venda: O modelo é composto pelos agentes de compra e venda que representam respectivamente o comprador e o vendedor. Esta seção apresenta os componentes de cada agente, detalhando as sua funcionalidades.

Protocolo de Negociação: Nesta Seção apresentamos o protocolo proposto para o funcionamento dos mecanismos de negociação utilizados no modelo. Cada fase do nosso protocolo será detalhado nessa Seção.

Trabalhos Relacionados: Os trabalhos relacionados a nossa pesquisa são abordados nesta seção. Um comparativo de cada trabalho com o nosso é comentado.

5.2 Diagrama de Transição de Estado do Modelo

O diagrama de estado do nosso modelo é bastante semelhante ao diagrama ilustrado na Figura 4.1. A principal diferença com o modelo original é a inclusão da transição *Call for*

Proposals no lugar da transição *request*, presente no modelo original para a inicialização da negociação. A outra diferença presente nesse modelo é a inclusão da transição *Call for Proposals* para abertura de uma nova sessão nos estados terminais *rejected* e *timeout*. Nesses dois estados terminais de falha, o consumidor pode optar por abrir uma nova sessão de negociaçao, caso a primeira nao venha satisfazer aos seus requisitos. Detalhes quanto à reabertura da nova sessão de negociação poderá ser obtida na Seção 5.5.5.

Call for Proposals é uma mensagem *multicast* do agente de compra para todos os agentes de venda que se encontram disponíveis para negociar em um determinado *marketplace* (local onde os agentes de venda possuem ofertas de produtos). A finalidade do *Call for Proposals* é a chamada por propostas e a mesma não implica compromisso algum por parte do agente chamador, porém esta transição faz com que os agentes de venda retornem com *propose, offer* ou até mesmo com *reject*. O novo diagrama de estado é ilustrado na Figura 5.1.

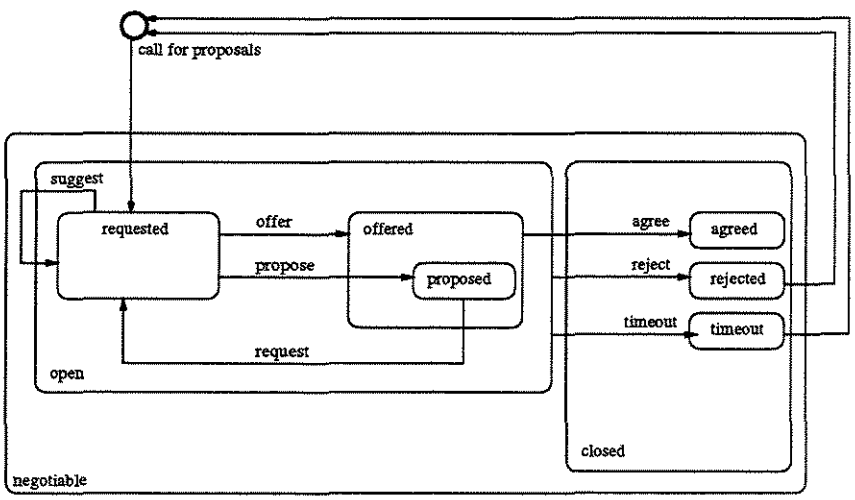


Figura 5.1: Diagrama de Transição de Estado do Modelo

5.3 Medidas de Similaridades

Esta Seção apresenta o mecanismo para determinar o produto mais similar ao produto requisitado pelo consumidor, entre os diversos que foram ofertados. O referido mecanismo é utilizado quando o produto requisitado pelo consumidor não for encontrado no catálogo.

A nossa abordagem é baseada nos pesos que são atribuídos a cada informação. Conforme ilustrado na Seção 6, o formulário de entrada de dados contém vários campos de

informação que por sua vez são ligados a um objeto *Slider* onde os pesos (de 1 a 10) são informados.

A medida de similaridade é calculada pela soma de todos os pesos elevados ao quadrado. Porém, o peso só é adicionado nos atributos cujas informações são as mesmas do valor do produto encontrado no catálogo. Se as informações não forem as mesmas, nenhum valor é adicionado. O processo descrito pode ser formalizado através da Equação 5.1.

$$W_T = \sum_{i=0}^{n} w_i^2 x_i \tag{5.1}$$

onde W_T é o valor da similaridade final; w_i é o valor peso do atributo i; x_i corresponde ao valor 0 ou 1 (do atributo i), dependendo se o valor encontrado for igual ao valor requisitado; e n é o número de atributos considerados na pesquisa. Quanto maior o valor de W_T, mais similar é o produto com o que foi requisitado. Vale salientar que a similaridade foi calculada utilizando o mecanismo descrito através desta equação, no intuito de prover uma ênfase maior para os maiores pesos atribuídos pelo consumidor.

5.3.1 Questão do w_i^2

Os pesos são elevados ao quadrado com o intuito de enfatizar os maiores valores, levando a similaridade para a característica mais relevante do consumidor. Existem muitas discussões sobre o fato de elevar-se ao cubo ou mesmo para valores mais altos. No entanto, diante das nossas simulações encontramos produtos mais condizentes com o que foi requisitado pelo consumidor ao elevar-se o peso ao quadrado. Caso os pesos fossem somados com os seus valores absolutos (sem elevar-se ao quadrado), os produtos seriam selecionados obedecendo-se uma ênfase uniforme para os valores dos pesos informados pelo consumidor. Em contrapartida o peso elevado ao quadrado provê uma seleção que dá uma ênfase maior aos valores dos pesos atribuídos para cada característica. Vale salientar que a presente discussão, assim como a nossa fórmula de similaridade, expressa um caso simples de otimização inteira.

O peso com valor 10 é um caso especial, o mesmo é informado com a finalidade de determinar que aquela propriedade não é negociável e deve ser igual ao valor requisitado. Por exemplo, se o consumidor deseja adquirir um carro e informa 10 como peso e o valor do atributo como *Honda Civic*, então o único valor aceitável para este atributo será *Honda Civic*. Um exemplo do uso da medida de similaridade será apresentado a seguir.

5.3.2 Exemplo

Suponha que o consumidor esteja procurando por um carro novo modelo *Honda Civic*, cor *azul*, com *ar condicionado, CD player, alarme* e *air bag*. Os pesos informados para cada atributo estão descritos na Tabela 5.1.

Atributo	*Valor*	*Peso*
Modelo	Honda Civic	9
Cor	Azul	4
Novo	Sim	6
Usado	Sim	4
Air Conditioned	Sim	8
CD Player	Sim	5
Alarme	Sim	6
Air Bag	Sim	5

Tabela 5.1: Atributos e valores informados pelo consumidor

Atributo	*Valor*	w_i^2	
Modelo	Honda Civic	9^2	81
Cor	Amarelo	0	0
Novo	Sim	6^2	36
Usado	Não	0	0
Ar Condicionado	Não	0	0
CD Player	Não	0	0
Alarme	Sim	6^2	36
Air Bag	Não	0	0

Atributo	*Valor*	w_i^2	
Modelo	Nissan Sentra	0	0
Cor	Preto	0	0
Novo	Sim	6^2	36
Usado	Não	0	0
Ar Condicionado	Não	0	0
CD Player	Sim	5^2	25
Alarme	Sim	6^2	36
Air Bag	Sim	5^2	25

Tabela 5.2: Veículos similares encontrados com (a) $W_T = 153$, e (b) $W_T = 122$

As Tabelas 5.2.a e 5.2.b apresentam veículos similares ao requisitado com os seus respectivos pesos elevados ao quadrado. A Tabela 5.2.a descreve um veículo mais similar do que o veículo na Tabela 5.2.b, de acordo com o maior valor de W_T. A primeira ocorrência possui um veículo mais similar, uma vez que ela representa um veículo contendo um número maior de atributos relevantes para o consumidor. Neste exemplo, os atributos mais relevantes são: modelo (*Honda Civic*) e *ar condicionado*. Ambos são mais importantes para o consumidor, tendo em vista que os mesmos foram informados com um peso

maior. Note que nenhum destes valores estão presentes na Tabela 5.2.b. Portanto, nesse exemplo, a proposta selecionada seria a primeira em função do maior valor de W_T.

É importante observar que a soma dos pesos (w_i) na Tabela 5.2.a (21) é menor que a soma na Tabela 5.2.b (22), porém a soma dos quadrados é maior na Tabela 5.2.a (153) do que na Tabela 5.2.b (122). Fica claro a partir deste exemplo que elevando-se o peso ao quadrado é possível enfatizar os maiores pesos atribuídos pelo consumidor.

5.4 Agentes de Compra e Venda

A idéia básica dos agentes de venda e de compra, é projetar o comportamento real de um consumidor e de um vendedor no nosso modelo. Assim, o agente de compra representa o consumidor em busca de algum produto, enquanto que os agentes de venda representam os diversos vendedores com os quais os consumidores interagem de forma a adquirir o produto que atenda às suas necessidades. A seguir iremos detalhar o projeto de cada um dos componentes.

5.4.1 Agente de Compra

O agente de compra representa o consumidor e no nosso modelo é responsável por barganhar o produto com o agente de venda. O agente de compra deve ser móvel em virtude de sua migração aos *hosts* dos agentes de venda. O referido agente é composto basicamente de dois componentes, conforme ilustrado na Figura 5.2.

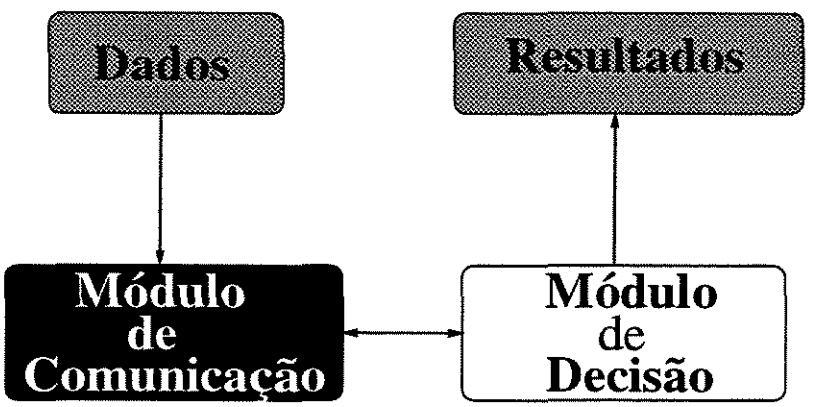


Figura 5.2: Componentes do Agente Móvel de Compra

módulo de Decisão: No agente original, o módulo de decisão diz respeito ao componente responsável pela aceitação ou rejeição de cada oferta proposta. Este componente

cuida também da seleção dos agentes de venda cujas ofertas serão negociadas. Para a seleção, o critério utilizado é a similaridade, assim como o preço ofertado. Nas cópias dos agentes, o presente módulo cuida da decisão quanto à continuação da barganha de preço durante a fase de Negociação.

módulo de Comunicação: O presente módulo é responsável por realizar a comunicação com os agentes de venda além de cuidar da comunicação com todas as cópias migradas para os *hosts* dos agentes de venda selecionados. O referido módulo é também responsável por armazenar as informações das ofertas durante o processo de comunicação e migração aos *hosts* dos agentes de venda.

5.4.2 Agente de Venda

O agente de venda no nosso modelo representa o vendedor e é responsável para oferecer propostas que atendam aos requisitos do consumidor. Além das funcionalidades básicas presentes também no agente de compra, o agente de venda possui funcionalidades relacionadas à leitura dos catálogos implementados em XML. Os componentes que pertencem ao agente de venda são ilustrados na Figura 5.3.

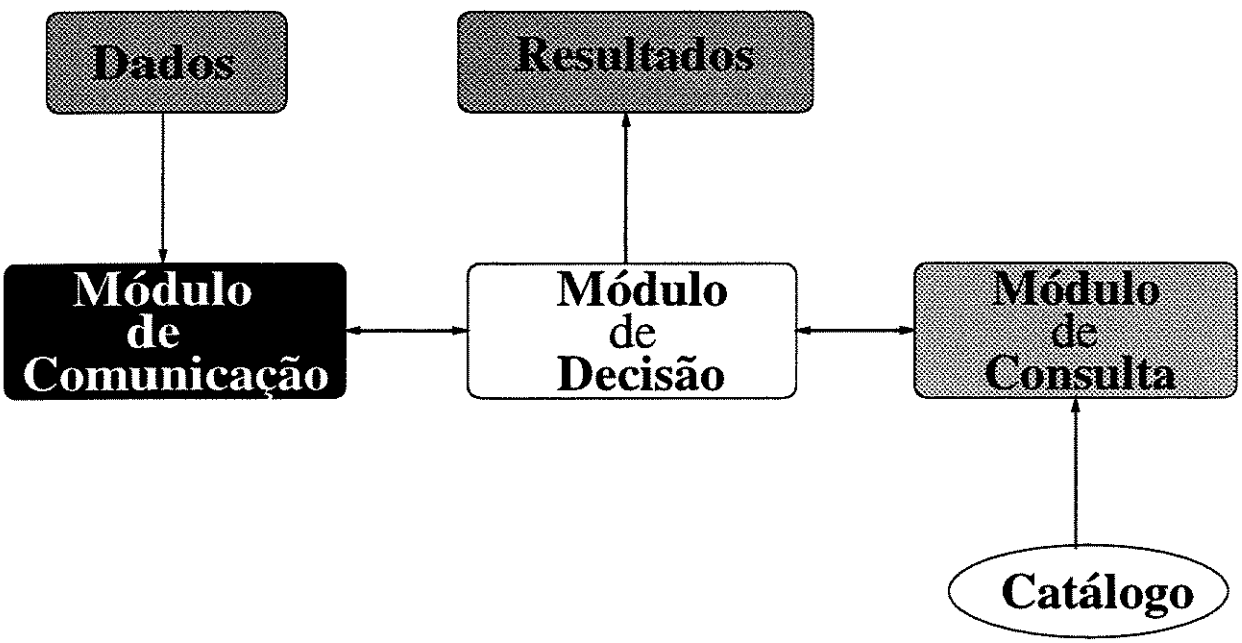


Figura 5.3: Componentes do Agente Móvel de Venda

Módulo de Comunicação: Este módulo é responsável para estabelecer a comunicação com o agente de compra.

Módulo de Decisão: O agente de venda pode também negar a negociação caso o consumidor proponha uma oferta muito distante do preço do seu catálogo. Esta decisão fica a cargo deste módulo que além disso tem a função também de ofertar uma proposta negociável (através do *propose*) ou uma oferta final ao consumidor (utilizando a transição *offer*), caso a negociação atinja o preço mínimo desejado pelo vendedor.

Módulo de Consulta: O módulo de consulta é responsável por procurar, ler e devolver a informação que foi requisitada pelo agente de compra. Em virtude da implementação da leitura ter sido codificada utilizando a API DOM, este módulo percorre a árvore criada pela API até que a informação desejada seja encontrada. O módulo de Consulta devolve a informação ao módulo de Decisão que toma uma decisão a seguir sobre a informação lida, devolvendo o seu resultado ao agente de compra que a solicitou.

5.5 Protocolo de Negociação

O nosso modelo possui duas modalidades de procura: *fixa* e *negociável*. Além disso, o protocolo é dividido em cinco fases (*chamada, seleção, negociação, apresentação* e *conclusão*), cada uma delas contendo funcionalidades diferentes [36, 37].

A procura *fixa* citada anteriormente é utilizada quando o consumidor deseja receber propostas de produtos com os mesmos atributos requisitados pelo consumidor. Portanto, essa pesquisa é mais utilizada quando o comprador está à procura de um produto específico, como é o caso da compra de um determinado livro. Os mecanismos utilizados para determinar as similaridades não são utilizados, uma vez que o consumidor não está interessado em produtos similares. Caso o mesmo não seja encontrado, uma mensagem é exibida informando o consumidor.

A outra modalidade de pesquisa (*negociável*) é uma extensão da primeira. Nesta modalidade, o consumidor informa o produto a ser adquirido que por sua vez é procurado em vários catálogos do *marketplace*. Porém, se o mesmo não for encontrado, um produto similar será pesquisado utilizando a métrica da similaridade, de forma a satisfazer o consumidor. Nesta modalidade, pelo menos uma sessão de negociação é aberta de forma a permitir a barganha de preço entre o comprador e o vendedor. A seguir detalharemos cada fase pertencente ao protocolo.

5.5.1 Fase da Chamada

A primeira fase diz respeito ao envio de uma mensagem *multicast* do agente de compra para todos os agentes de venda com a finalidade de invocar por propostas. O agente de compra pode consultar o serviço de *Trader* [24] ou algum outro similar para encontrar agentes de venda apropriados. A presente fase é formalizada através da transição *call for proposals* ilustrada na Figura 5.1.

Após a chamada por propostas, o agente de venda pode responder com: *propose*, *offer* ou *reject*. *Propose* e *offer* são utilizadas para enviar proposta para o consumidor. Por outro lado, *reject* pode ser utilizada para rejeitar o pedido do cliente em virtude do mesmo estar negociando com um número excessivo de agentes de compra ou até mesmo em função do agente de venda não vislumbrar um bom negócio nessa negociação. Por exemplo, se o agente de compra propõe um preço com um valor bastante distante do preço desejado pelo vendedor, o agente de venda nega a sua participação na negociação. No nosso modelo, o agente de venda nega a participação na negociação, caso a primeira proposta seja inferior a 20% do preço estabelecido no catálogo do vendedor.

Nesta fase, o agente de compra pode chamar o *Timeout* caso o agente de venda demore em responder à chamada por propostas em virtude do mesmo se encontrar bastante ocupado com outras negociações.

A Figura 5.4 ilustra o cenário proposto para a primeira fase onde um agente de compra envia uma chamada por propostas para quatro agentes de venda. Nesse exemplo, apenas o *SellAgent1* rejeita o pedido do cliente, negando a sua participação na negociação, enquanto que *SellAgent2*, *SellAgent3* e *SellAgent4* aceitam negociar, respondendo para isso com *offer* ou *propose*. A diferença entre as duas transições, conforme foi explicado anteriormente, é de que o primeiro indica uma proposta final enquanto que a última é uma oferta que é negociável.

5.5.2 Fase da Seleção

Após o recebimento das propostas na fase anterior, o agente de compra está agora apto a selecionar os agentes de venda com quem irá interagir. O critério utilizado para selecionar é a proposta enviada. Se o agente de compra vislumbrar um mau negócio, o mesmo invoca o *reject* de forma que possa negar a negociação com esse agente. A previsão de uma negociação com um ganho desfavorável pode vir de dois fatos:

- preço desejado muito distante do preço ofertado.
- número de atributos iguais do produto ofertado e do requisitado está aquém do que foi pré-estabelecido.

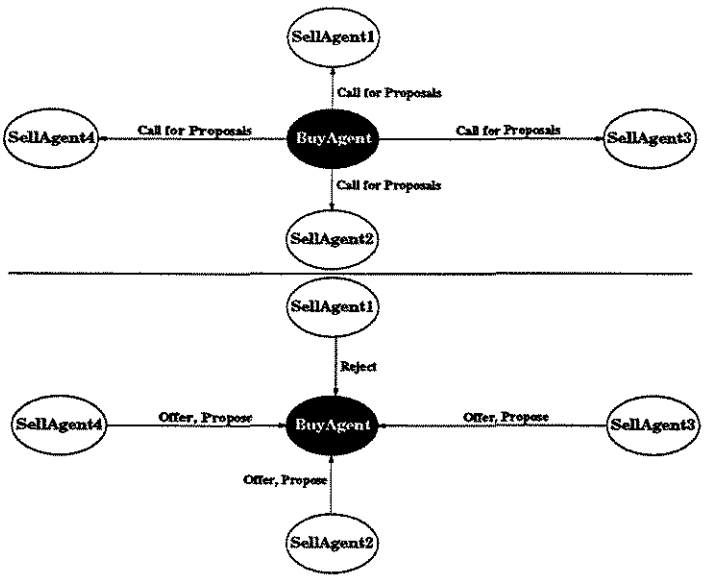


Figura 5.4: Cenário proposto para a fase de Chamada

A seleção dos agentes de venda é formalizada através da transição *request* que na realidade é uma contraproposta da chamada lançada na primeira fase (*Chamada*). Se o agente de venda responder a chamada através da transição *offer* e caso o agente de compra vislumbre um ganho favorável nessa proposta, então a mesma será armazenada com a finalidade de ser apresentada ao consumidor juntamente com outras propostas na fase da *Apresentação.*

A seleção dos produtos é realizada através do cálculo da similaridade abordado anteriormente. No nosso modelo, cada agente de venda calcula o W_T para cada produto e manda ao agente de compra como proposta o que prover o maior valor do W_T. O preço do produto é negociado na próxima fase (negociação).

Na Figura 5.5, apenas o *SellAgent3* e o *SellAgent4* foram selecionados para negociar com o agente de compra, posto que o mesmo não vislumbrou um ganho muito bom a partir da negociação com o *SellAgent2.*

5.5.3 Fase da Negociação

Após a seleção dos agentes, o agente de compra está agora apto a negociar com os mesmos de forma a alcançar o melhor preço. Os mecanismos de barganha de preço nesse modelo são baseados no *Kasbah.* Vale salientar que a negociação neste presente trabalho se limita apenas à barganha de preço.

Antes da negociação propriamente dita, o agente de compra é copiado para os *hosts*

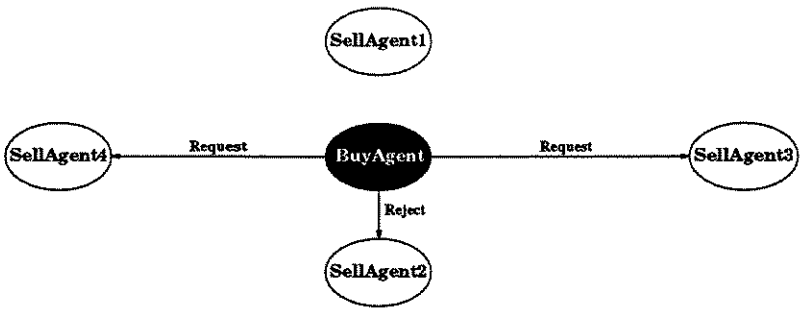


Figura 5.5: Cenário proposto para a fase de Seleção

onde cada agente de venda selecionado está localizado. A cópia e a mobilidade para as respectivas máquinas são vantajosas principalmente em virtude da negociação envolver uma quantidade de mensagens muito grande, ocasionada pelas propostas e contrapropostas entre os agentes. A cópia pode evitar a comunicação na rede, reduzindo o tráfego na rede.

A Figura 5.6 ilustra as funcionalidades e as transições presentes durante a fase de *negociação*. Os agentes de venda e de compra ofertam propostas e contrapropostas através das transições *offer* e *propose*. Por outro lado, as transições *request* e *suggest* são utilizadas para invocar por novas propostas.

A sessão da negociação é fechada diante de três eventos:

- o agente de venda oferta uma proposta final através da transição *offer*;
- chamada do *timeout* em virtude da ausência de proposta durante um tempo pré-estabelecido.
- o agente de venda oferta o preço desejado pelo consumidor.

A estratégia do agente de venda para obter um maior ganho será iniciar a negociação com um preço acima do desejado para venda e diminuir até o seu preço mínimo estabelecido pelo vendedor. De forma similar, o agente de compra poderá iniciar ofertando propostas com preços abaixo do desejado pelo consumidor e aumentar o mesmo durante o processo de negociação até que se alcance o preço máximo estabelecido.

Como trabalho futuro pretendemos desenvolver um modelo que possa barganhar outros atributos além do próprio preço, que utilize também nesta fase a métrica de similaridade.

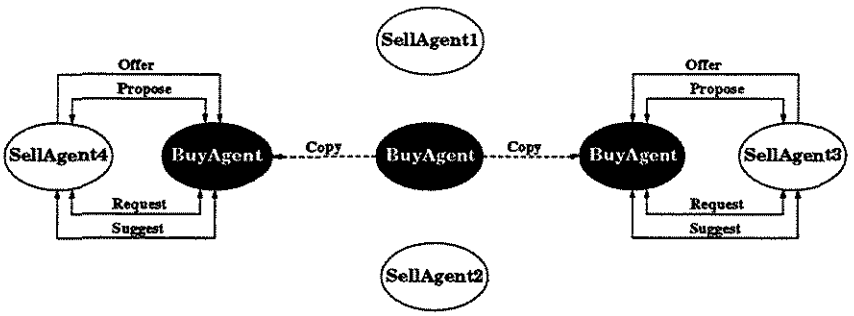


Figura 5.6: Cenário proposto para a fase de Negociação

5.5.4 Fase da Apresentação

Após a conclusão da negociação de preço executada entre cada par de agente de venda e de compra, cada cópia do agente comprador apresenta os resultados alcançados na sessão de negociação. Em outras palavras, na fase da *apresentação* (Figura 5.7) cada cópia do agente de compra envia uma proposta resultante da negociação para a sua cópia original. Estas propostas são apresentadas ao consumidor que poderá aceitar ou negar cada proposta. Após a entrega das propostas, cada cópia do agente de compra pode ser removida do *host* para onde a mesma foi copiada.

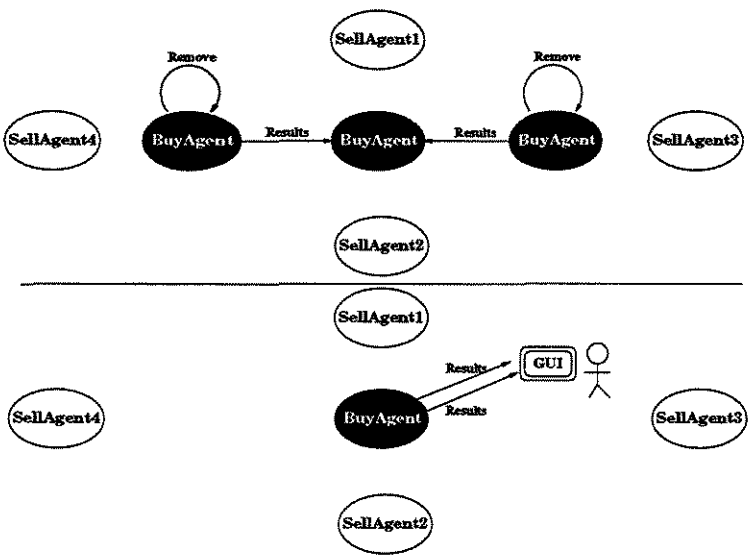


Figura 5.7: Cenário proposto para a fase de Apresentação

5.5.5 Fase da Conclusão

A fase da *conclusão* diz respeito ao fechamento da sessão de negociação. A negociação é concluída através da formalização de uma das respostas: *agree* ou *reject.*

É válido ressaltar que o resultado final é informado pelo consumidor. A Figura 5.8 ilustra a última fase do protocolo de negociação. Nesta Figura a proposta do *SellAgent3* é rejeitada, enquanto que o do *SellAgent4* é aceita pelo consumidor.

Neste modelo o consumidor pode também abrir uma nova sessão atribuindo-se novos valores aos pesos ou alterando-se os atributos fornecidos anteriormente. Esse processo pode ser repetido até que o consumidor encontre o produto que possa satisfazê-lo, caso não encontre um exatamente com os atributos informados.

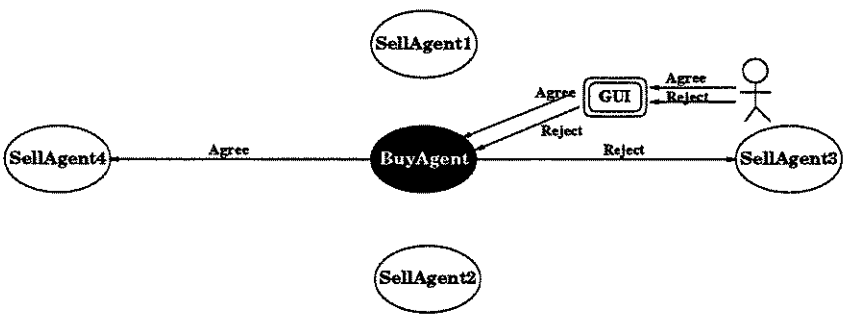


Figura 5.8: Cenário proposto para a fase de Conclusão

5.6 Trabalhos Relacionados

Existem vários trabalhos desenvolvidos na área de negociação em comércio eletrônico, porém grande parte da pesquisa nessa área é focalizada para solucionar os mecanismos de leilão como é o caso do *OFFER* (*Object Framework for Electronic Requisitioning*) [5], que trata da negociação entre um grupo de participantes, possuindo mecanismos para corretagem (*brokerage*) para Internet. O principal objetivo deste trabalho é implementar um sistema de corretagem aberto que seja capaz de ser operacionalizado em várias áreas de negócio. O mecanismo utilizado para realizar a negociação nesse modelo é o leilão.

A principal diferença deste trabalho em comparação ao nosso é o mecanismo de negociação. Enquanto no *OFFER* utiliza-se os mecanismos de leilão, o nosso trabalho realiza a negociação de uma forma bilateral (consumidor e fornecedor). Além desta diferença, *OFFER* enfatiza mais a questão da corretagem aberta, enquanto que o presente trabalho trata mais da negociação entre um consumidor e um vendedor.

Além deste trabalho, outro que merece destaque é o modelo de leilão descentralizado (*Dependable Distributed Auction System*), onde não existe um elemento central que controla o seu funcionamento. O referido modelo aborda um leilão que é considerado como distribuído [10]. Atualmente, os leilões eletrônicos são realizados na sua grande maioria no ambiente da Internet, que possui uma abrangência global. Este trabalho enfatiza a implementação dos mercados locais nos ambientes de leilão. A idéia básica deste trabalho é estabelecer vários mercados locais, cada um deles contendo as políticas pertencentes a cada país ou região. Dentre as políticas pertencentes a cada mercado, podemos citar os aspectos monetários de cada país ou região. Por sua vez, cada um desses mercados de caráter local são sujeitos (dependentes) aos regulementos mais abrangentes, pertencentes ao mercado como um todo (global).

Da mesma forma que o *OFFER*, a nossa pesquisa não modela o mecanismo de leilão, além disso o trabalho proposto em [10] trata mais da implementação de um leilão distribuído e descentralizado com a finalidade de se criar vários mercados locais contendo as políticas locais, porém todos eles ligados a um mercado global, provendo uma maior escalabilidade nesses ambientes de leilão.

Outro trabalho na área negociação similar ao nosso utilizando a abordagem da similaridade (*Using Similarity Cryteria to Make Negotiation Trade-Offs*) pode ser encontrado em [11]. Este trabalho utiliza a notação *fuzzy* para determinar as similaridades e assim poder ofertar propostas que sejam condizentes com a realidade de ambas as partes. O objetivo principal deste trabalho é propor um algorítmo baseado na lógica *fuzzy* para realizar barganhas.

A nossa pesquisa é bastante semelhante com o trabalho citado em [11] no que diz respeito ao uso da similaridade para realizar barganhas. Porém é válido ressaltar que o trabalho descrito em [11] enfatiza mais a apresentação do algorítmo para realizar barganhas. Em contrapartida, a nossa pesquisa vai além disso, ao propor um mecanismo para determinar as similaridades, além do próprio protocolo de negociação bilateral.

Outros trabalhos existentes utilizam os mecanismos de aprendizado com objetivo de prover a melhor estratégia de barganha. Estes mecanismos obtiveram resultados satisfatórios para solucionar certos problemas, porém a sua maior desvantagem é o tempo decorrido para o seu aprendizado e a complexidade gerada para desenvolver este modelo. O uso dos algoritmos genéticos proposto em [23] requer cerca de 20 a 4000 gerações para obter uma boa estratégia de negociação, segundo um estudo em [2].

Finalmente, um trabalho que merece atenção é o *Kasbah* [6]. Este trabalho desenvolvido pelo MIT Media Laboratory apresenta um protótipo para negociar de forma automatizada o preço do produto. Nesse modelo os agentes de compra e venda barganham até que os dois entrem em acordo quanto ao preço do produto. O algorítmo de barganha não utiliza os métodos de aprendizado, porém nesse modelo os agentes de compra possuem

capacidade de armazenar informações sobre o produto que o consumidor deseja adquirir. Da mesma forma, os agentes de venda contêm informações relevantes para a venda do produto. Essas informações incluem o preço mínimo requisitado pelo fornecedor, assim como o preço desejado para a venda.

A nossa pesquisa utiliza as funcionalidades de barganha de preço existente no *Kasbah*. Além dos mecanismos de barganha de preço, o nosso trabalho possui os mecanismos para determinar as similaridades entre os produtos. Um protocolo de negociação também foi proposto no nosso trabalho com a finalidade de prover um ambiente de negociação. Da mesma forma que no *Kasbah*, implementamos um protótipo para validar os conceitos citados neste trabalho.

Capítulo 6

Implementação do Modelo de Negociação Automatizada

6.1 Visão Geral

O modelo foi implementado em Java utilizando o JDK versão 1.2.2 (*Java Development Kit*) e o sistema de agentes utilizado foi o *Grasshopper*. Esta plataforma de agentes foi adotada, em virtude da mesma possuir a interface MASIF (*Mobile Agent System Interoperability Facility*) [25]. Esta interface é importante para o modelo, uma vez que ela provê a interoperabilidade entre sistemas de agentes diferentes. Como o nosso modelo é voltado para aplicações de comércio eletrônico, a interoperabilidade torna-se um requisito indispensável. Os catálogos foram escritos em XML e a interface gráfica utilizada pelos usuários foi codificada em Java *Swing* que é parte do JFC (*Java Foundation Class*), podendo ser utilizada junto com o *JDK 1.2.2*.

6.2 Protótipo

Com finalidade de validar o modelo proposto, um protótipo foi desenvolvido para aplicações de compra e venda de automóveis. As Figuras 6.1 e 6.2 ilustram os formulários para pesquisa *fixa* e pesquisa *negociável*, respectivamente.

Os formulários ilustrados dizem respeito ao contexto de compra e venda de automóveis. A pesquisa *fixa*, conforme mencionada anteriormente, não possui os mecanismos de negociação. Por outro lado, a *negociável* permite a negociação. Ambos os formulários permitem o preenchimentos dos dados do veículo desejado pelo consumidor, como modelo, cor, ano (de fabricação e de modelo), tipo (usado ou novo), os opcionais do veículo, assim como o limite máximo tolerável para a entrega do automóvel.

Na procura *negociável*, além desses dados, o consumidor deve informar os pesos referentes a cada informação fornecida, de forma que o modelo possa calcular a similaridade, caso o produto exato não seja encontrado no catálogo. Ainda nessa modalidade de pesquisa, o consumidor deve informar o preço desejado e o máximo que ele está disposto a pagar. O ano mínimo e o máximo limitam a pesquisa para o intervalo informado. Finalmente, a função de elevação de preço determina a política de elevação dos preços no decorrer do tempo. As três opções existentes no formulário correspondem aos três gráficos ilustrados na Figura 6.9.

É importante ressaltar que os formulários ilustrados podem apresentar uma mudança no seu conteúdo dependendo do ambiente de execução e/ou perfil do usuário. Por exemplo, caso este modelo de negociação seja executado em ambientes móveis onde existam poucos recursos computacionais, é indispensável que o formulário exibido nesse ambiente seja menos carregado no intuito de prover um desempenho compatível com este ambiente.

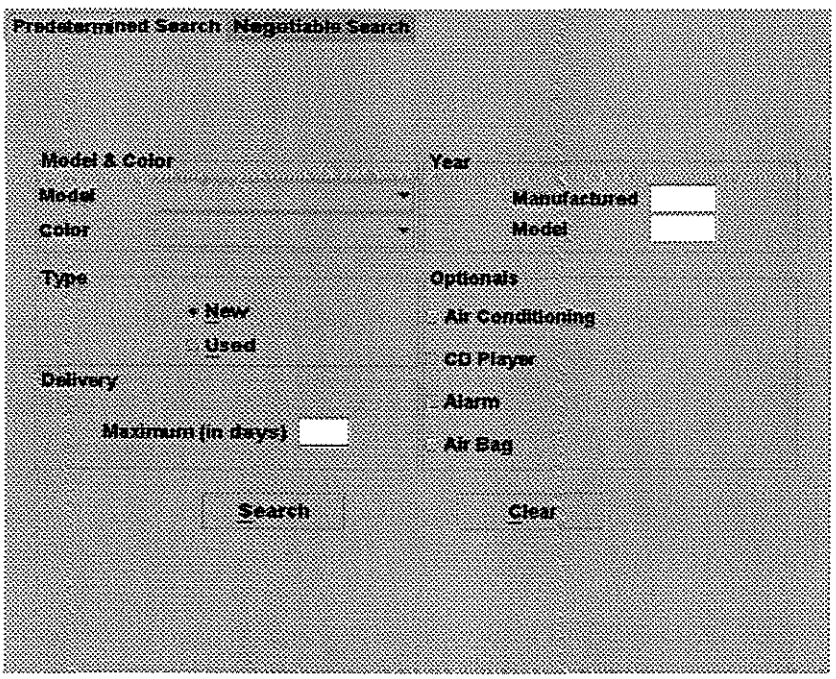


Figura 6.1: Formulário utilizado para pesquisa *fixa*

6.2.1 Agente de Compra

O agente de compra conforme descrito no Capítulo 5 diz respeito ao agente móvel que representa o consumidor, portanto possuindo a função de adquirir produtos especificados

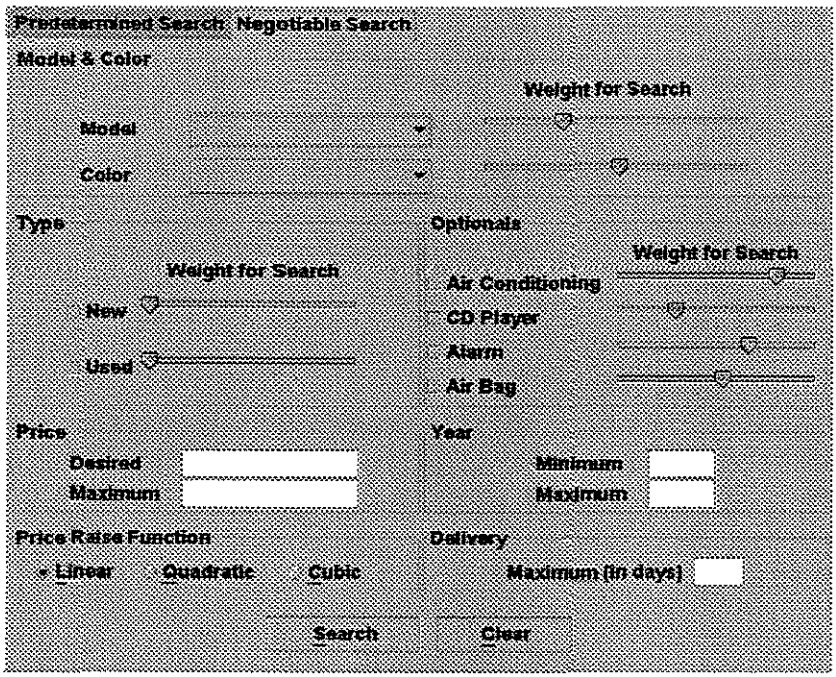


Figura 6.2: Formulário utilizado para pesquisa *negociável*

por ele. Esse agente deve ser móvel em virtude do mesmo migrar para o *hosts* dos agentes vendedores. É válido ressaltar que este comportamento foi implementado no nosso modelo de forma que os agentes (de compra e de venda) pudessem possuir o mesmo comportamento existente no mundo real. Por exemplo, o consumidor normalmente vai a várias lojas especializadas com o intuito de adquirir o produto com a melhor oferta.

O agente de compra foi implementado a partir da classe *MobileAgent* que é uma subclasse da classe *de.ikv.grasshopper.agent*. A classe *MobileAgent* provê vários métodos necessários para a mobilidade do agente.

6.2.2 Agente de Venda

No nosso modelo o vendedor é representado pelos agentes de venda. Vale salientar que diferente do comprador que é representado por apenas um agente de compra e pelas respectivas cópias, o vendedor é representado por vários agentes móveis distintos. A função do agente de venda é ofertar o produto que mais se assemelha ao que foi requisitado pelo consumidor.

Assim como no agente de compra, o de venda também foi implementado a partir da classe *MobileAgent*, subclasse de *de.ikv.grasshopper.agent*.

6.2.3 Plataforma Grasshopper

A comunicação entre o agente de venda e de compra é baseado no *serviço de comunicação* disponível no *Grasshopper*. No nosso protótipo, o agente de venda é executado como se fosse o servidor que está a espera de requisições dos clientes. Os clientes seriam os agentes de compra que chamam métodos do agente de venda (servidor) para realizar a chamada por propostas, assim como para ofertar e requisitar propostas.

Existem vários tipos de comunicação implementados no *Grasshopper*. O mecanismo assíncrono foi utilizado para evitar que o agente de compra ficasse bloqueado durante o período em que o servidor estivesse processando a requisição do cliente. Assim o agente de compra original processaria cada oferta à medida que as propostas fossem enviadas pelas cópias migradas para os *hosts* do agente de venda.

6.2.4 Implementação das Fases

Fase da Chamada

No protótipo construído, a *chamada por propostas* é enviada a todos os agentes de venda que fazem parte de uma mesma *região*. Porém caso as regiões possuíssem agentes de venda ofertando diversos produtos que não sejam apenas automóveis, uma consulta ao *Trader* é necessário para filtrar os agentes de venda de seu interesse.

Portanto no nosso protótipo, os agentes de venda podem estar em *agências* separadas, desde que façam parte de uma mesma *região* para que os mesmos recebam a chamada. A partir desta premissa, é possível afirmar que o *marketplace* do nosso protótipo é composto por todos os agentes de venda que pertencem a uma determinada *região*. A Figura 6.3 ilustra a implementação da fase da Chamada do nosso modelo.

Fase da Seleção

Conforme foi abordado no Capítulo 5, esta fase diz respeito à etapa onde o agente de compra seleciona os agentes de venda que irão interagir. O método *propose* é utilizado para confirmar a seleção, enquanto que *reject* indica que o agente foi rejeitado. No nosso protótipo, o agente de compra seleciona apenas os agentes de venda cujo valor de W_T seja igual ou superior à metade do W_T das características requisitadas pelo consumidor. Por exemplo, para os pesos e características fornecidas pelo consumidor na Tabela 5.1 (W_T=299), o agente de compra do nosso protótipo só selecionaria o agente de compra que propusesse a oferta descrita na Tabela 5.2(a), uma vez que a metade do valor total do W_T nesse exemplo totaliza 149,50 que é o limite mínimo do W_T para que o agente de compra do nosso protótipo possa selecionar os agentes de venda. Vale salientar que a oferta descrita na Tabela ??(b) não seria selecionado pelo protótipo, em virtude da mesma

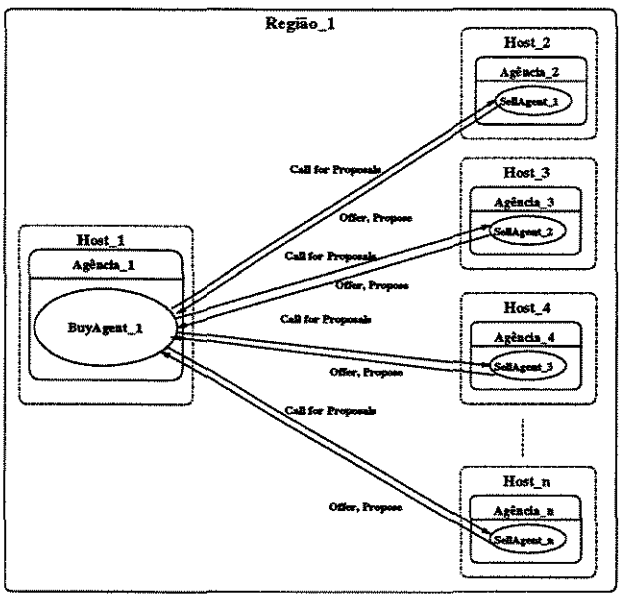


Figura 6.3: Implementação da Chamada por Propostas

não atingir o valor mínimo. A Figura 6.4 ilustra a implementação da fase da Seleção no nosso protótipo, utilizando a plataforma *Grasshopper*.

Fase da Negociação

Após a seleção dos agentes de venda, o agente de compra é copiado para os *hosts* dos agentes de venda. Essa cópia é executada através do método *copy()* existente na plataforma *Grasshopper*. No protótipo construído, os preços propostos por ambas as partes (pelo agente de venda e pelo agente de compra) são gerados a partir de um gráfico linear onde o eixo vertical corresponde ao valor do preço a ser ofertado e o eixo horizontal ao tempo decorrido. O gráfico utilizado no protótipo pelo agente de venda é o *linear* (ilustrado na Figura 6.9). Porém é válido ressaltar que o modelo permite a seleção de um dos gráficos (*linear, quadrática* ou *cúbica* para determinar a política de oferta de preço. A Figura 6.5 apresenta a estrutura do protótipo desenvolvido para a fase de Negociação do nosso modelo.

No protótipo desenvolvido para o contexto de compra e venda de automóveis, o agente de compra oferta inicialmente dez por cento inferior ao preço desejado pelo consumidor, elevando essa oferta à medida do tempo até atingir o preço máximo desejado. O preço máximo assim como o desejado é informado pelo consumidor no formulário ilustrado na Figura 6.2. A Equação 6.1 ilustra a fórmula utilizada para calcular o preço a ser

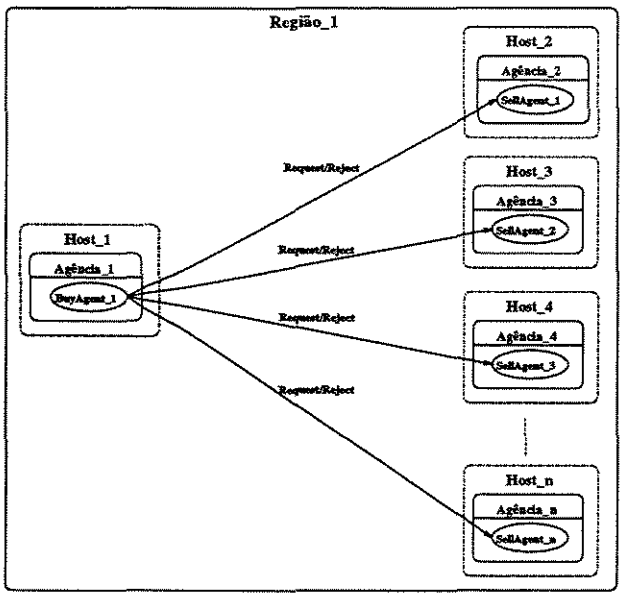


Figura 6.4: Implementação da Fase da Seleção

ofertado pelo agente de compra, onde *priceRequested* é o próximo valor a ser proposto, *java.lang.System.currentTimeMillis()* é a variável do sistema que armazena o tempo atual em milisegundos e *initialTime* é a variável que armazena o tempo em que o agente iniciou a negociação, ou seja a diferença (*java.lang.System.currentTimeMillis()* - *initialTime*) resulta no tempo decorrido até o momento em que a oferta será proposta. A variável *initialPrice* armazena o valor da primeira oferta calculada.

Em contrapartida, o agente de venda, inicialmente oferta o preço sugerido no catálogo, decrescendo o seu valor até atingir o mínimo requerido pelo fornecedor. No nosso protótipo, consideramos que o fornecedor sugere inicialmente dez por cento acima do desejado, decrescendo esse valor à medida do tempo, conforme o gráfico ilustrado na Figura 6.9. A Equação 6.2 descreve a fórmula utilizada para calcular o preço a ser ofertado pelo agente de venda. A variável *priceOffered* armazena o valor da próxima oferta, *catalogPrice* contém o valor do preço sugerido no catálogo e *initialTime* indica o tempo em que o agente de venda iniciou a negociação.

$$priceRequested = java.lang.System.currentTimeMillis() - initialTime + initialPrice; \quad (6.1)$$

$$priceOffered = catalogPrice - (java.lang.System.currentTimeMillis() - initialTime); \quad (6.2)$$

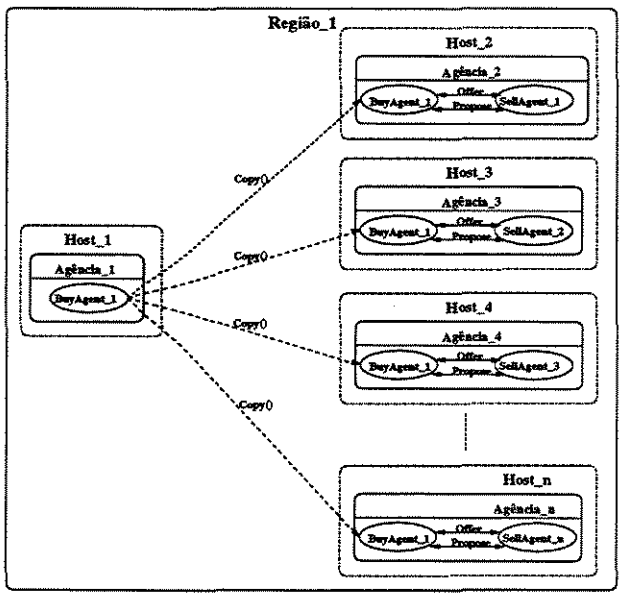


Figura 6.5: Implementação da Fase de Negociação

Fase da Apresentação

Esta fase, conforme abordada no Capítulo 5, apresenta os resultados obtidos durante a fase de negociação, onde cada cópia do agente de compra envia a oferta final à sua cópia original. O preço final negociado é enviado através de um vetor global que armazena todas as ofertas finais. A exclusão das cópias é efetuada através do método *remove()* pertencente à classe *de.ikv.grasshopper.agent*. A Figura 6.6 ilustra a implementação da fase de Apresentação do modelo.

Fase da Conclusão

Na fase da Conclusão o agente de compra recebe as instruções do consumidor de forma que ele possa selecionar as ofertas e rejeitar as outras que não sejam de seu interesse. No protótipo construído, cada oferta é associada a um agente de venda através de um identificador que foi implementado através da classe *Identifier*, presente na plataforma *Grasshopper*. O identificador único é utilizado para notificar o *agree* ou o *reject* para cada

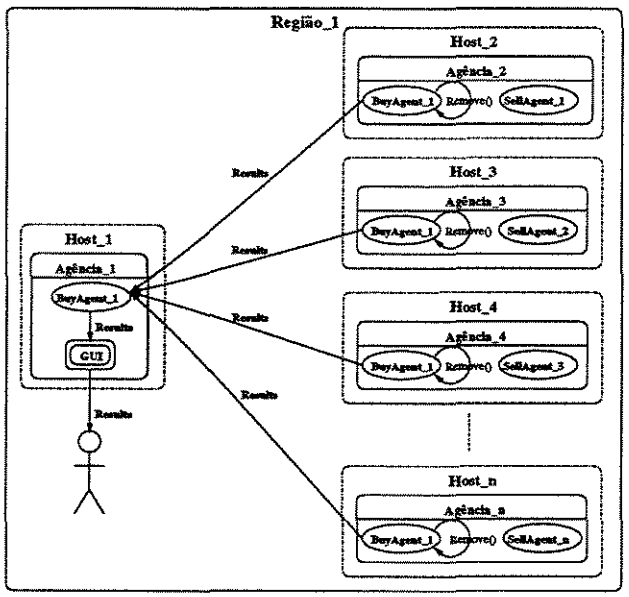


Figura 6.6: Implementação da Fase de Apresentação

agente de venda que propôs a oferta. A Figura 6.7 apresenta a implementação da fase de Conclusão na plataforma *Grasshopper*.

6.2.5 Interface de Comunicação entre os Agentes

Cada transição presente no modelo é chamada invocando-se um método da interface *SellAgent* (Figura 6.8).

Métodos Implementados na Interface

CallForProposals

O método *CallForProposals* é utilizado pelo agente de compra para chamar pelas propostas e possui como parâmetro o vetor que armazena os atributos do veículo a ser encontrado, retornando outro vetor que possui as propriedades do veículo disponível no catálogo que mais se assemelha ao veículo requisitado.

Request e Suggest

Os métodos *request* e *suggest* são utilizados para requerer por novas sugestões. No caso do *request* ela pode ser utilizada também para transitar de um estado com maior grau de compromisso para um com menor. Ambos os métodos possuem como parâmetro a classe *Identifier*, implementada no sistema *Grasshopper* para identificar o agente a quem

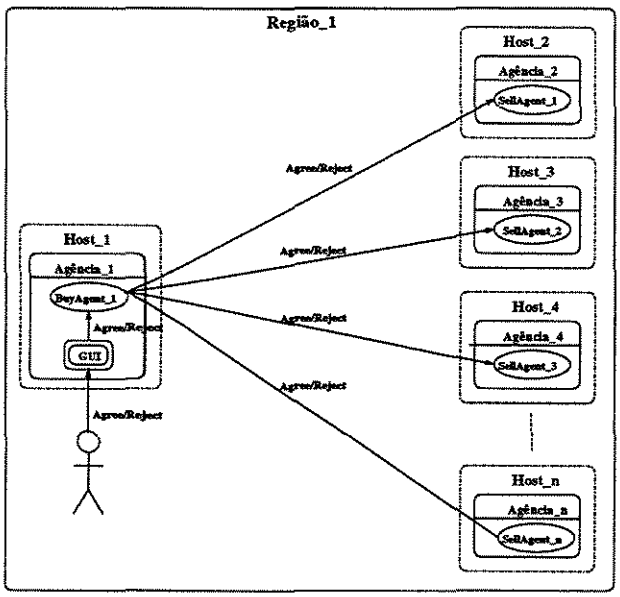


Figura 6.7: Implementação da Fase de Conclusão

se destina a mensagem. *Suggest* e *request* retornam um *String* usado para certificar de que a transição foi bem sucedida.

Propose e Offer

Propose e *Offer* são utilizados na fase da *negociação* para ofertar propostas e contrapropostas. No protótipo implementado, *Propose* é utilizado também para formalizar a seleção dos agentes de venda que serão interagidos. Ambos os métodos passam como parâmetros o identificador e o preço sugerido. *Propose* retorna os endereços onde os agentes de venda estão localizados, uma vez que esses endereços são necessários para informar os *hosts* para onde as cópias dos agentes de compra serão migrados.

Agree e Reject

Agree e *Reject* foram implementados para aceitar ou rejeitar a proposta ofertada. Os referidos métodos possuem como parâmetro os identificadores de agente e retornam um *String* que confirma a execução da operação chamada.

6.2.6 Geração das Ofertas de Preço

No modelo implementado o vendedor pode selecionar uma das três estratégias de geração dos preços a serem ofertados: *linear, quadrática* e *cúbica.* Da mesma forma, o comprador

```
package BuySell;

import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.*;

public interface SellAgent
{
    public String[] CallForProposals(String[] vetarg);

    public String suggest(Identifier agentId);

    public String request(Identifier agentId);

    public String agree(Identifier agentId);

    public String reject(Identifier agentId);

    public GrasshopperAddress propose(Identifier agentIdent,long pricebidden);

    public String offer(Identifier agentIdent,long pricebidden);
}
```

Figura 6.8: Métodos utilizados para invocar as transições

pode também selecionar um dos mecanismos para elevar o preço a ser ofertado. A Figura 6.9 ilustra os gráficos da geração das ofertas para o vendedor. Por outro lado, o gráfico do comprador é semelhante ao apresentado na Figura 6.9, porém com o valor do preço sendo elevado durante o decorrer do tempo [6].

No nosso modelo, o ponto de encontro entre a função do comprador (que eleva o preço no decorrer do tempo) e a função do vendedor (que reduz o preço) indica o acordo na negociação. Evidentemente que o acordo entre ambos também se concretiza, caso a função do vendedor oferte um preço abaixo do valor proposto pela função do comprador. Da mesma forma, o acordo se concretiza caso a função do comprador oferte um valor acima do que foi proposto pela função do vendedor.

6.2.7 Cópia e Exclusão dos Agentes

A cópia do agente de compra para o *host* do agente de venda é realizada apenas para a barganha de preço, uma vez que a troca de mensagens é bem maior durante esta fase. A cópia

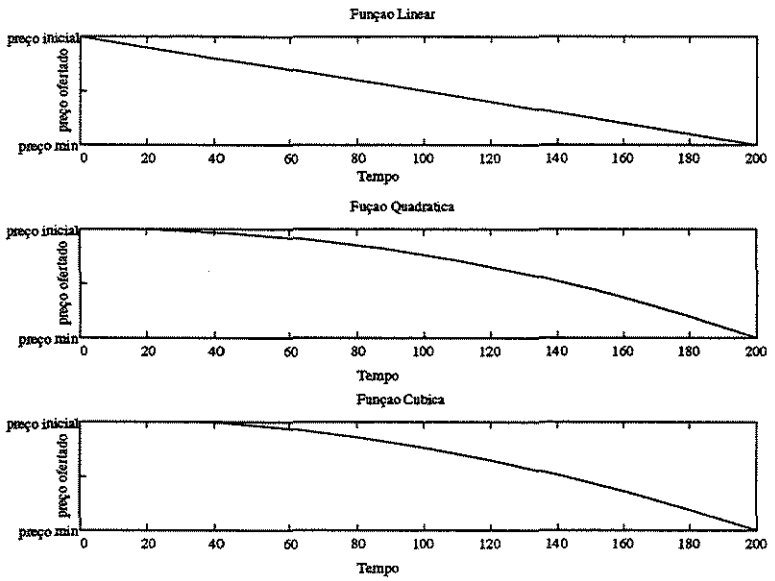


Figura 6.9: Gráfico para geração do preço a ser proposto pelo vendedor

é realizada através do método *copy()* existente no *Grasshopper*. Este método necessita do endereço de destino que no modelo seria o *host* do vendedor. O *Grasshopper* também provê o método *beforeCopy()* que é utilizado para codificar as instruções referentes ao armazenamento temporário dos atributos do automóvel, até que complete a barganha de preço. A exclusão das cópias é realizada através do método *remove()* disponível também no *Grasshopper*.

6.2.8 Catálogos

Os catálogos foram implementados em XML com a finalidade de prover a interoperabilidade necessária nos sistemas de comércio eletrônico. Para executar o protótipo, foi construído um pequeno catálogo para armazenar os automóveis disponíveis para a venda. A Figura 6.10 apresenta parte do catálogo usado no protótipo.

A integração entre Java e XML é implementada no nosso modelo, através do uso da API (*Application Program Interface*) DOM (*Document Object Model*) [18] que transforma um documento XML em uma árvore de objetos. A API SAX (*Simple API for XML*) [18] foi avaliada como uma API alternativa, porém para implementar o protótipo utilizamos o DOM, em virtude da mesma ser uma API mais completa, além de ser recomendada pela W3C.

No catálogo desenvolvido, implementamos a restrição espacial, ou seja, a restrição

```
<?xml version = "1.0"?>
<AUTOMOBILES>
  <AUTOMOBILE>
    <MODEL>HONDA CIVIC EX</MODEL>
    <TYPE>USED</TYPE>
    <YEAR>1999</YEAR>
    <COLOR>BLUE</COLOR>
    <PRICE>9000</PRICE>
    <DELIVERY>0</DELIVERY>
    <OPTIONAL>
      <AIRCOND>YES</AIRCOND>
      <CD>NO</CD>
      <ALARM>NO</ALARM>
      <AIRBAG>YES</AIRBAG>
    </OPTIONAL>
  </AUTOMOBILE>
  <AUTOMOBILE>
    <MODEL>HONDA ACORD EX</MODEL>
    <TYPE>NEW</TYPE>
    <YEAR>2001</YEAR>
    <COLOR>WHITE</COLOR>
    <PRICE>25000</PRICE>
    <DELIVERY>0</DELIVERY>
    <OPTIONAL>
      <AIRCOND>YES</AIRCOND>
      <CD>YES</CD>
      <ALARM>YES</ALARM>
      <AIRBAG>YES</AIRBAG>
    </OPTIONAL>
  </AUTOMOBILE>
<AUTOMOBILES>
```

Figura 6.10: Parte do catálogo utilizado no protótipo

que determina a abrangência espacial do catálogo. Por exemplo, é interessante dispor certos catálogos de produtos apenas para alguns países ou continentes, restringindo a sua abrangência. No nosso protótipo, disponibilizamos algumas *regiões* de agência para restringir espacialmente os catálogos.

Capítulo 7

Conclusão

O presente trabalho contribuiu com uma proposta de modelo para comércio eletrônico baseado em um protocolo de negociação automatizada e em métricas de similaridades, além das simulações do uso de XML para catálogos eletrônicos.

Uma das vantagens deste modelo é o paralelismo na negociação. Uma vez que cada cópia do agente de compra é copiado para o *host* do agente de venda, negociações autônomas e paralelas podem ser operacionalizadas, otimizando bastante o tempo de procura do consumidor por um determinado produto. Em virtude das cópias serem autônomas, diferentes ofertas poderão ser propostas, ampliando-se o repertório de opções de compra do consumidor. A nossa idéia com o presente modelo de negociação é implementar vários compradores que operacionalizam as suas funções de compra para um consumidor apenas. No mundo real podemos representar este cenário como um determinado comprador que designa vários subordinados a procurarem por ofertas de um produto que deseja adquirir. Estes subordinados iriam pesquisar e barganhar por ofertas, trazendo a melhor oferta ao seu superior que tem o papel de consumidor, que por sua vez seleciona a melhor entre elas para efetuar a compra.

A outra contribuição deste trabalho diz respeito à medida de similaridade. O mecanismo apresentado não se baseia em bases de dados, onde é armazenada uma grande quantidade de informações que poderá ajudar na determinação das similaridades. Além disso, a mesma também não é baseada nos mecanismos de aprendizado, porém a métrica proposta é determinada de forma simples, bastando que o usuário indique o seu perfil de preferência, através dos valores dos pesos requisitados.

Para propor o uso de XML no desenvolvimento dos catálogos, implementamos simulações com os catálogos locais e remotos. Neste último, realizamos comparações das tecnologias de comunicação remota (RMI, CORBA e agentes móveis *Grasshopper*) mais comumente utilizadas pelas aplicações para acessar os catálogos remotos. Além deste estudo, realizamos comparações entre as duas APIs mais utilizadas no mercado para in-

tegrar Java e XML. Nesse estudo ressaltamos as vantagens e as desvantagens no uso de cada API.

No presente trabalho implementamos o nosso modelo utilizando os agentes móveis em decorrência dos resultados obtidos na simulação e da avaliação geral do uso das tecnologias de comunicação, descritos no Capítulo 3. A partir destas simulações verificamos que o uso dos agentes móveis não é adequado para aplicações que façam apenas uma única leitura nos catálogos remotos. Isso se justifica, em virtude do tempo de latência necessário para a migração dos agentes para os *hosts* onde estão localizados os catálogos. Em função deste resultado, concluímos que o uso dos agentes móveis seria mais vantajoso no nosso modelo, tendo em vista a grande quantidade de propostas e contrapropostas a serem transmitidas na rede. Com a migração dos agentes de compra para os *hosts* dos agentes de venda não ocuparíamos a rede com as respectivas propostas e contrapropostas.

A plataforma *Grasshopper* foi utilizada em virtude da interface MASIF, presente nesta plataforma. Além disso, *Grasshopper* permite o uso livre desta plataforma, desde que se cadastre a finalidade do seu uso. Durante o desenvolvimento do protótipo, pudemos também verificar a grande quantidade de funcionalidades existentes na mesma, como a comunicação assíncrona que foi implementada para a troca de mensagens entre o agente de venda e de compra.

A interoperabilidade é alcançada através do uso de *XML* na confecção dos catálogos, assim como na implementação em Java para prover um código portável entre arquiteturas diferentes. Porém é válido ressaltar que a interoperabilidade dos catálogos só poderá ser alcançada mediante o uso dos padrões de comunicação descritos na Seção 3.2.

Das simulações efetuadas no Capítulo 3, podemos concluir que CORBA é a melhor escolha para ler catálogos remotos em XML, posto que o mesmo apresentou tempos bem próximos de RMI. Além disso, CORBA possui serviços e facilidades que poderão ser úteis na implementação de aplicações de comércio eletrônico, como é o caso do serviço de *Trader*. Apesar destas vantagens, implementamos o nosso modelo utilizando os agentes móveis em virtude do mesmo ser mais vantajoso em aplicações de negociação com uma exaustiva quantidade de propostas e contrapropostas trafegadas pela rede. A partir das simulações presentes também no mesmo capítulo, podemos concluir que a API SAX é a melhor escolha na implementação de códigos utilizados para acessar catálogos em XML. SAX apresentou melhores desempenhos quanto ao tempo de acesso, além de ter requerido menos código em comparação à DOM.

Como trabalho futuro pretendemos estender o uso da medida da similaridade também na fase da negociação, permitindo que propostas e contrapropostas possam ser ofertadas a partir do uso dessas medidas, até que o protocolo encontre uma métrica que seja mais adequada frente à requisição do consumidor. Uma outra extensão seria permitir a comunicação entre as cópias dos agentes de compra, de forma que os mesmos possam

notificar a melhor oferta alcançada entre eles. Dessa forma, barganhas desnecessárias seriam evitadas, poupando o tempo de negociação.

Uma outra idéia que estamos investigando diz respeito à abrangência dos agentes a serem interagidos. Pretendemos que o nosso protocolo tenha acesso a outros domínios de serviços gerenciados por outros sistemas de agentes. Esta idéia é possível de ser implementada, posto que o *Grasshopper* possui interface *MASIF* que provê a interoperabilidade entre diferentes plataformas de agentes. Um outro mecanismo que concluímos ser importante para o bom desempenho deste protocolo é a inserção de um mecanismo que limite o tempo máximo de negociação.

Consideramos também como trabalho futuro a modelagem e a verificação de propriedades nas diversas fases do protocolo. Um outro trabalho que vale a pena ser salientado diz respeito à modelagem do nosso protocolo de negociação para que o mesmo seja executado em ambientes de comércio eletrônico móvel (*m-commerce*).

Referências Bibliográficas

[1] ActiveMedia. Web Revenues. `http://www.atctivemedia.com/press/webrevenues.htm`.

[2] C. Beam e A. Segev. Electronic catalogs and negotiations, Agosto de 1996. CITM Working Paper 96-WP-1016, Fisher Center for Information Technology and Management, Universidade da California em Berkeley.

[3] C. Beam e A. Segev. Automated Negotiations: A Survey of the State of the Art, 1998. CITM Working Paper 98-WP-1022, Fisher Center for Information Technology and Management, Universidade da California em Berkeley.

[4] C. Beam e A. Segev. CMIT Program on Automated Bargaining and Negotiation in Electronic Commerce, 1999. The Fisher Center for Management and Information Technology, University of California em Berkeley.

[5] M. Bichler, C. Beam, e A. Segev. OFFER: An object framework for electronic requisitioning, Dezembro, 1997. CITM Working Paper 97-WP-1026, Fisher Center for Information Technology and Management, University da California em Berkeley.

[6] A. Chavez e P. Maes. Kasbah: An agent marketplace for buying and selling goods. Nos Proceedings do First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), pp.75-90, 1996.

[7] CommerceNet. eCo Framework. `http://www.commerce.net`.

[8] G. M. Doss. *XML Developer´s Guide For CORBA*. Wordware Publishing, 1999.

[9] OMG e CommerceNet. The OMG/CommerceNet Join Eletronic Commerce Whitepaper, Julho de 1997.

[10] P. Ezhilchechelvan e G. Morgan. A Dependable Distributed Auction System: Architecture and an Implementation Framework. Nos Proceedings do Fifth IEEE International Symposium on Autonomous Descentralized Systems (ISADS'01), pp.03-10, Dallas, EUA, Março de 2001.

[11] P. Faratin, C. Sierra, e N. R. Jennings. Using Similarity Criteria to Make Negotiation Trade-offs. Nos Proceedings do Fourth Int. Conf. on Multi-Agent Systems (ICMAS-2000), pp.199-126, Boston, EUA, 2000.

[12] L. C. Ferreira. Sistemas de Pagamento Eletrônico: Classificação, Análise e Implementação. Dissertação de Mestrado. UNICAMP - Universidade Estadual de Campinas, Novembro de 1998.

[13] P. Flynn. Frequently Asked Questions About the Extensible Markup Language. `http://www.ucc.ie/xml`, Junho de 1999.

[14] IBM. Aglets Mobile Agent System. `http://www.aglets.org`, 2001.

[15] N. Idris. SAX Tutorial 1. `http://www.developerlife.com`.

[16] N. Idris. Should I use SAX or DOM? `http://www.developerlife.com`.

[17] IKV. Grasshopper - The agent platform, 1999. Users' Guide.

[18] JavaWorld. XML for the Absolute Beginner: a Guide Tour from HTML to Processing XML with Java, Abril de 1999.

[19] B. Marchal. *XML By Example.* QUE Publishing, 1999.

[20] Martsoft. IntuiCat application suite. `http://www.martsoft.com/products/index.html#INTUICAT`.

[21] H. Maruyama, K. Tamura, e N. Uramoto. *XML and Java: developing Web applications.* Addison Wesley Publishing, 1999.

[22] ObjectSpace. Voyager Agent System. `http://www.objectspace.com`, 2001.

[23] J. Oliver. A machine learning approach to automated negotiation and prospects for electronic commerce. *Journal of Management Information Systems*, 13(3):83–112, Winter 1996-97.

[24] OMG. Trading Object Service Specification, 1997.

[25] OMG. MASIF (Mobile Agent System Interoperability Facility), 1998.

[26] OMG. Negotiation Facility - Final Revised Submission, Março de 1999.

[27] OMG. Object Management Group. `http://www.omg.org`, 1999.

[28] OMG. E-Commerce and the OMG (white paper), Fevereiro de 2000.

[29] R. Orfali e D. Harkey. *Client/Server Programming with Java and CORBA.* Wiley Computer Publishing, 1998.

[30] Planetcommerce. Comércio Eletrônico - Conceitos, Modelos e Arquiteturas. `http://www.planetcommerce.com.br/comercio.htm`, 1999.

[31] Tech. Requisite. Technology and Services. `http://www.requisite.com`.

[32] E. J. Rodrigues. Uma Modelagem para Comércio Eletrônico usando CORBA e Agentes Móveis. Dissertação de Mestrado. UNICAMP - Universidade Estadual de Campinas, Fevereiro de 1999.

[33] Cons. RosettaNet. Página Oficial. `http://www.rosettanet.com`.

[34] T. W. Sandholm. An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. Nos Proceedings do 12th International Workshop on Distributed Artificial Intelligence, Hidden Valley, Pennsylvania, pp.295-308, 1993.

[35] R. G. Smith. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. Nos IEEE Trans. on Computers, pp.1104-1113, volume 12, capítulo 29, 1980.

[36] J. Ueyama e E. R. M. Madeira. An Automated Negotiation Model for Electronic Commerce. Nos Proceedings do Fifth IEEE International Symposium on Autonomous Decetralized Systems - ISADS'01, pp.29-36, Dallas, EUA, Março de 2001.

[37] J. Ueyama e E. R. M. Madeira. Um Modelo de Negociação para Comércio Eletrônico. Nos Proceedings do Simpósio Brasileiro de Redes de Computadores - SBRC'2001, pp.653-668, Florianópolis, Brasil, Maio de 2001.

[38] J. Ueyama e E. R. M. Madeira. Using XML for Electronic Catalogs. Nos Proceedings do International Workshop on Information Integration on the Web - WIIW 2001, pp.43-50, Rio de Janeiro, Brasil, Abril de 2001.

[39] J. Ueyama e E. R. M. Madeira. Aplicação de Catálogos XML para Comércio Eletrônico. *Developer's Magazine*, (46):20–28, Junho de 2000.

[40] W3C. The World Wide Web Consortium. `http://www.w3.org/XML/`.

[41] D. Zeng e K. Sycara. Bayesian Learning in Negotiation. Em Working Notes do (AAAI) Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems, Editora Sandip Sen, pp.99-104, Stanford University, CA, EUA, 1996.