UNIVERSIDADE FEDERAL DO MARANHÃO

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

ÁREA DE CIÊNCIA DA COMPUTAÇÃO

SÉRGIO GOMES MARTINS

MODELO DE NEGOCIAÇÃO E O AGENTE MEDIADOR NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

São Luís

2007

MODELO DE NEGOCIAÇÃO E O AGENTE MEDIADOR NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

SÉRGIO GOMES MARTINS

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Eletricidade na área de Ciência da Computação.

Orientador: Prof. Dr. Sofiane Labidi

São Luís

2007

Martins, Sérgio Gomes.
Modelo de negociação e o agente mediador no ambiente ICS de comércio eletrônico / Sérgio Gomes Martins. – 2007.
111f.

Impresso por computador (fotocópia)
Orientador: Sofiane Labidi
Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, São Luís, 2007.

1. Comércio eletrônico. 2. ICS – modelo de negociação. 3. Comércio eletrônico – agente mediador - protótipo. I. Labidi, Sofiane, orient. II. Título.

CDU 004.735.5:339.9

SÉRGIO GOMES MARTINS

MODELO DE NEGOCIAÇÃO E O AGENTE MEDIADOR NO AMBIENTE ICS DE COMÉRCIO ELETRÔNICO

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Eletricidade na área de Ciência da Computação.

Dissertação aprovada em 14 de Setembro de 2007

BANCA EXAMINADORA

_____

Prof. Sofiane Labidi, Dr.
(Orientador)

_____

Prof. Pedro Porfírio Muniz Farias, Dr.

_____

Prof. Zair Abdelouahab, Ph. D.

A Deus pela minha existência.

"Há duas fontes perenes de alegria pura: o bem realizado e o dever cumprido."

(Eduardo Girão)

AGRADECIMENTOS

Aos meus pais, pelo apoio, pela minha formação e pelos bons princípios que me repassaram.

À minha esposa, Idalina, e aos meus filhos, Lucas e Maria Luíza, pelo carinho e pelo amor que tornam a minha vida cada dia mais feliz.

Ao meu orientador, professor Sofiane Labidi, pelo apoio e pelo estímulo para a realização deste trabalho.

Ao colega Bernardo Maia Jr., que contribuiu valorosamente para o desenvolvimento deste trabalho.

E a todos àqueles que, de alguma forma, me auxiliaram nesta conquista.

RESUMO

Este trabalho é parte do Projeto ICS (*Intelligent Commerce System*), atualmente em desenvolvimento no Laboratório de Sistemas Inteligentes (LSI), da Universidade Federal do Maranhão (UFMA), sob a coordenação do Prof. Dr. Sofiane Labidi. O projeto tem como objetivo desenvolver um ambiente inteligente de negociação de comércio eletrônico entre empresas (*business-to-business*). O ICS é baseado na tecnologia de agentes inteligentes móveis e numa abordagem em ciclo de vida composto de cinco fases. Este trabalho foca a característica principal do ICS que é a automação do processo de negociação para proporcionar às empresas condições favoráveis de comercialização a um baixo custo operacional, com rapidez e eficiência. A negociação é a fase em que os agentes inteligentes que representam os usuários (empresas) iniciam suas buscas pela satisfação de suas necessidades através de trocas de propostas de compra ou venda de bens e serviços. Apresentamos o modelo de negociação proposto e o protótipo do agente mediador, responsável pelo gerenciamento do processo de negociação como um todo.

Palavras-chave: Comércio Eletrônico, Agentes de *Software*, Modelo de Negociação, Agente Mediador.

ABSTRACT

This work is part of the ICS Project (Intelligent Commerce System), currently in development in the Intelligent Systems Laboratory (LSI), at Universidade Federal do Maranhão (UFMA), under the coordination of the Prof. Dr. Sofiane Labidi. The project aims at developing an Intelligent environment of negotiation for electronic commerce among companies (business-to-business) and in a approaching in cycle of life composed of five stages. This work aims at main feature of the ICS that is the automation of the negotiation process to provide companies with favorable conditions for commercialization at a low operational cost, with rapidity and efficiency. The negotiation is the stage where the intelligent agents who represent the users (companies), initiate its searches for the satisfaction of its necessities through exchanges of purchase or seller proposals of goods and services. We present the proposed Negotiation Model and a prototype of the mediator agent which is responsible for the management of the negotiation process as a whole.

Keywords: E-commerce, Software Agents, Negotiation Model, Mediator Agent.

SUMÁRIO

LISTA DE TABELAS

LISTAS DE FIGURAS

ABREVIATURAS E SÍMBOLOS

ICS	Intelligent Commerce System
B2B	Business to Business
EDI	Electronic Data Interchange
SMA	Sistemas Multiagentes
AOSE	Engineering Systems of Software Agents
UML	Unified Modeling Language
AUML	Agent Unified Modeling Language
MESSAGE	Methodology for Engineering Systems of Software Agents
FIPA	Foundation for Intelligent Physical Agents
OMG	Object Management Group
MIT	Massachusetts Institute of Technology
SMACE	Sistema Multiagente para Comércio Eletrônico
CAAT	Collaborative Agreement for Automatic Trading
HP Lab	Hewllet-Packard Laboratories
JADE	Java Agent Development Framework
JESS	Java Expert System Shell
URI	Uniform Resource Identifier
OWL	Web Ontology Language
TILAB	Telecom Italia Labs
ACL	Agent Communication Language
API	Application Programming Interface
SOMA	Security and Open Mobile Agents
SNL	Sandia National Laboratories

1 INTRODUÇÃO

1.1 Justificativa

O comércio eletrônico (*e-commerce*) é uma das áreas que mais crescem com o uso da internet, no mundo e também no Brasil. A procura e a oferta de bens e serviços de modo relativamente fácil e barato, num mercado virtualmente globalizado, possuem atrativos bastante óbvios para quase todas as empresas que desejam fazer negócios. Segundo dados de pesquisa realizada pela eBit[1] e compilada pela eCommerceOrg[2] somente no biênio 2005-2006, houve no Brasil um aumento de 76% nas vendas *on-line* totalizando um faturamento de R$ 4,40 bilhões no ano de 2006, como pode ser observado na Tabela 1.

Tabela 1: Faturamento anual das vendas *on-line* no Brasil.

ANO	FATURAMENTO	VARIAÇÃO
2007 Previsão	R$ 6,40 bilhões	45%
2006	R$ 4,40 bilhões	76%
2005	R$ 2,50 bilhões	43%
2004	R$ 1,75 bilhões	48%
2003	R$ 1,18 bilhões	39%
2002	R$ 0,85 bilhões	55%
2001	R$ 0,54 bilhões	-

Por outro lado, a tecnologia de agentes de *software*, um tipo de programa inteligente capaz de agir sem intervenção humana ou de outros sistemas, já atingiu um estágio de desenvolvimento em que podem representar ou auxiliar pessoas na execução de diversas tarefas. Os agentes podem realizar suas funções ininterruptamente, 24 horas por dia, 7 dias por semana, com confiabilidade e eficiência.

[1] Portal de compras *on-line* reunindo diversas empresas (www.ebit.com.br).
[2] *Site* governamental sobre comércio eletrônico (www.e-commerce.org.br).

Unindo esses dois campos – o *e-commerce* e a tecnologia de agentes – o sistema de comércio eletrônico inteligente ICS (*Intelligent Commerce System*) tem o objetivo de prover um ambiente de negociação cujos agentes de *software* possam representar as empresas em busca de oportunidades de negócios, ou seja, fornecendo ou comprando bens e serviços.

Os agentes seriam criados por usuários nas empresas para ofertar ou adquirir esses bens e serviços. Teriam informações sobre as condições de fornecimento ou aquisição e também como e quando essas poderiam variar. De posse dessas informações, os agentes seguiriam para um mercado virtual no qual poderiam alcançar seus objetivos através da realização de uma negociação assistida pelo próprio sistema através de um agente mediador.

Toda a estrutura formada pelo ICS proporciona aos seus usuários condições favoráveis para a comercialização automatizada e a baixo custo operacional. Isso favorece aos usuários ganhos em produtividade, através da redução de custos e de uma maior agilidade no fechamento dos negócios, aumentando, assim, a competitividade entre eles.

Entendemos, então, que o ICS seja um sistema inovador, podendo se configurar como uma ferramenta de grande valor para a área de comércio eletrônico. Embora algumas etapas do seu ciclo de vida tenham sido delineadas em outros trabalhos, compreendemos ser necessário dar continuidade ao seu desenvolvimento, no sentido de demonstrar sua viabilidade e funcionalidade. Sendo assim, queremos contribuir para seu aperfeiçoamento através da proposição do modelo de negociação e do desenvolvimento de um protótipo do agente mediador, que se constituem como objetos deste estudo.

1.2 Objetivos

O objetivo deste trabalho é propor o modelo de negociação utilizado na fase de mesmo nome do ICS e, como conseqüência, desenvolver um protótipo do agente mediador, que servirá como gerenciador da negociação que ocorre entre os outros

agentes participantes do sistema. O ICS é um sistema de comércio eletrônico inteligente entre empresas (B2B – *business-to-business*) baseado em agentes de *software*, no qual esses representam as empresas em busca de realização de negócios.

Especificamente, pretende-se:

- Propor o modelo de negociação utilizado pelos agentes para a realização da negociação;
- Desenvolver um protótipo do agente mediador que irá lidar com a negociação;
- Propor a modelagem inicial da ontologia da negociação para o ambiente ICS;
- Propor um protocolo de negociação que poderá ser utilizado em negociações do tipo leilão inglês (leilão direto) dentro das especificidades necessárias ao desenvolvimento do protótipo do agente mediador.

1.3 Estruturação da Dissertação

A presente dissertação está estruturada em sete capítulos. O capítulo 1 se compõe desta Introdução, contendo os objetivos e a justificativa deste trabalho e a descrição de sua estrutura.

No capítulo 2, fazemos uma revisão bibliográfica, apresentando o estado da arte das tecnologias utilizadas neste trabalho. No capítulo 3, descrevemos o ambiente ICS, seu ciclo de vida e sua arquitetura. No capítulo 4, apresentamos o modelo de negociação proposto para o ICS, primeira parte de nossa contribuição. No capítulo 5 e 6, apresentamos a segunda parte de nossa contribuição, através da modelagem e da prototipagem do agente mediador respectivamente.

Finalmente, no capítulo 7 apresentamos as conclusões e sugestões de trabalhos futuros.

2 ESTADO DA ARTE

Neste capítulo, apresentamos o estado da arte das tecnologias utilizadas neste trabalho. Iniciamos abordando o comércio eletrônico, em seguida os sistemas multiagentes e concluímos com a engenharia de software baseada em agentes.

2.1 Comércio Eletrônico

O comércio eletrônico – ou *e-commerce*, como é mais conhecido – pode ser definido de várias formas, dependendo do que seja levado em consideração na definição. Sua definição mais básica é a de qualquer negócio cuja transação se dê de forma eletrônica, seja entre parceiros de negócio ou entre uma empresa e seus clientes (CAMERON, 1997).

Entretanto, além da transação, há de se levarem em conta, também, os aspectos que, embora não façam parte do comércio em si, também são relevantes para a sua realização ou continuação, como as propagandas, a divulgação de informações, o suporte pós-venda e até mesmo as tecnologias usadas na sua implementação.

Em Fonseca (2003, p. 18), encontramos uma definição que busca englobar todos esses aspectos do comércio eletrônico:

> toda transação que ocorre por meio digital e que envolve a troca de bens ou serviços, sendo as diferentes visões do seu conceito associadas à sua aplicabilidade, e geradas a partir da necessidade de melhorar interações com consumidores, aprimorar processos de negócios e trocar informações dentro e entre empresas e clientes, fazendo o melhor uso possível da Tecnologia da Informação.

O comércio eletrônico está constantemente evoluindo. Partiu de um estágio no qual as empresas parceiras de negócios podiam usar tecnologias como o *Electronic Data Interchange* (EDI) para automatizar alguns aspectos da realização de negócios, mas que dificultava a troca de parceiros, devido aos investimentos em infra-estrutura. E chegou a outro estágio, em que a *World Wide Web* se destacava

pela busca e oferta de novos parceiros de negócio através de *sites* que serviam como ponto de encontro entre eles. A desvantagem era que, com a diversidade de parceiros, tornava-se cada vez mais difícil automatizar a realização dos negócios em si, devido principalmente ao fato de as empresas possuírem diferentes dimensões e participarem de diferentes setores da economia.

Um exemplo desse segundo estágio de desenvolvimento do comércio eletrônico são os portais de concorrência. Eles são *sites* na *Web* que concentram a oferta ou a procura de bens ou serviços e intermedeiam as relações entre parceiros que desejam negociar, estabelecendo padrões comuns de valores, condições de negociação e pagamento, bem como informações. A maioria deles funciona na forma de leilões virtuais (no sentido de não serem presenciais) pela Internet e são bastante conhecidos como tais. Como exemplo, temos, no cenário mundial, o eBay[3], e, no âmbito nacional, seu *site* parceiro, o Mercado Livre[4].

Outro portal de concorrência que realmente faz jus a esse nome é o Comprasnet[5], portal do governo brasileiro que concentra as licitações – neste caso sob a forma de pregões – de órgãos de todas as esferas governamentais. Nesses pregões, o órgão governamental publica suas necessidades de bens ou serviços através de um edital e os fornecedores competem através de ofertas de preços para atender essas necessidades, numa espécie de leilão reverso (os “vendedores” fazem lances cada vez menores para o “comprador”). Um problema que todos esses *sites* possuem, no entanto, reside no fato de ser necessário que os usuários procurem, de forma mais ou menos manual (dependente dos serviços do portal), os negócios em que estejam interessados.

É então nesse estágio que a pesquisa sobre comércio eletrônico agora se concentra: como automatizar a busca e a realização de negócios, aumentando sua eficiência e velocidade, sem que a tecnologia ou a variedade de parceiros se constitua num obstáculo.

---

[3] www.ebay.com
[4] www.mercadolivre.com.br
[5] www.comprasnet.gov.br

Uma solução que parece atender aos requisitos desse problema são os sistemas baseados em agentes de *software*, assunto que será abordado a seguir.

2.2 Sistemas Multiagentes

A definição de agente de *software* encontra obstáculos, em razão de sua versatilidade. Diferentes definições aparecem, ao se destacarem diferentes características de um agente como sendo mais relevantes em determinados ambientes. Uma das definições mais aceitas é a de que “um agente é um sistema computacional que está situado em algum ambiente e que é capaz de executar ações autônomas de forma flexível neste ambiente, a fim de satisfazer seus objetivos de projeto” (WOOLDRIDGE et al., 1998, p. 279).

Podemos entendê-los também, de conformidade com a definição de Wooldridge, como entidades de *software* autônomas que interagem com seu ambiente, de tal forma, que o agente percebe o seu meio através de sensores e atua nele com seus efetuadores, como ilustrado na Figura 1 (OMG, 2000).

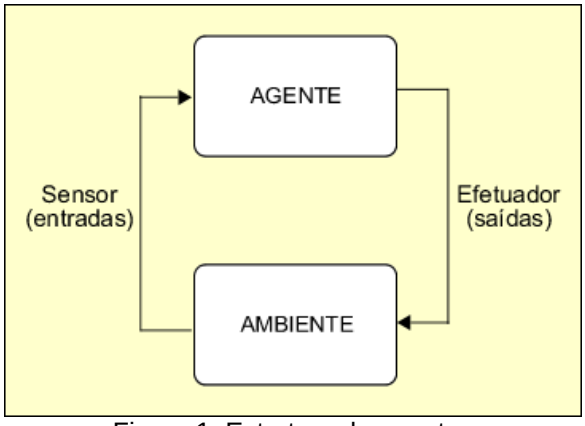

Figura 1: Estrutura de agentes

A autonomia dos agentes de *software* é a característica comum a todas as definições. Ela significa que os agentes devem decidir por si próprios que ações devem ser executadas para atingir seus objetivos ou metas. Essas ações levam em conta e podem modificar o ambiente em que o agente está situado.

Além da autonomia, outras características que os agentes devem possuir são:

- **Sociabilidade**: capacidade de interagir com outros agentes de software ou humanos para atingir seus objetivos ou auxiliar outros em suas atividades;
- **Racionalidade**: os agentes devem sempre agir no sentido de atingir suas metas, não devendo aceitar objetivos contraditórios ou que não sejam compensadores (para atingir as metas);
- **Adaptabilidade**: o agente deve ser capaz de se ajustar às características, preferências e hábitos do seu usuário;
- **Mobilidade**: o agente deve ser capaz de migrar para outros pontos da rede, de acordo com sua necessidade e conveniência;

Além dessas, outras características existem, mas elas dependem principalmente das aplicações nas quais os agentes são usados. A mobilidade, por exemplo, é essencial para os agentes negociantes do sistema ICS, os quais devem se deslocar para locais definidos, onde realizarão suas negociações, a fim de reduzir custos de comunicação e permitir diferentes protocolos de negociação. Já para o agente mediador, que deve interagir com os agentes negociantes para realizar o processo de negociação, a mobilidade não é muito importante, contudo a sociabilidade é imprescindível para que ele venha alcançar seus objetivos.

Agentes de *software* são desenvolvidos para trabalhar em conjunto, formando os Sistemas Multiagentes (SMA), sociedades de agentes que executam ações, comunicam-se e interagem para atingir seus próprios objetivos e os do sistema, coordenando-se através de cooperação ou de competição ou ainda na combinação de ambas as propriedades. Sistemas Multiagentes são, então, compostos por agentes coordenados através de seus relacionamentos com outros agentes, capacitando-os a resolver problemas que vão além da simples soma de suas capacidades individuais.

Algumas considerações são feitas pelo Grupo OMG a respeito dos sistemas multiagentes (OMG, 2000):

- Não se deve criar agentes que façam tudo sozinhos, pois dessa forma tais agentes representarão problemas de performance, confiabilidade, manutenção, etc. Dividindo as funcionalidades entre vários agentes, obtêm-se maiores modularidade, flexibilidade, extensibilidade e manutenção facilitada.

- O conhecimento que está espalhado em vários agentes pode ser integrado, para se obter uma visão geral, quando necessário.

- Aplicações que requerem processamento distribuído são mais bem suportadas por Sistemas Multiagentes. Para isso, os agentes podem ser projetados como pequenos componentes autônomos que trabalham em paralelo.

Muitas são as aplicações possíveis para os Sistemas Multiagentes, pois existem sistemas para várias áreas de aplicação: industrial, comercial, médica, etc. Alguns exemplos dessas aplicações são:

- Controle de tráfego – os agentes recebem informações de diversos sensores e, através da interpretação desses dados, estabelecem forma de melhorar o tráfego aéreo, por exemplo;

- Sistemas de transportes – os agentes representam os bens ou os usuários transportados;

- Jogos com personagens – os agentes podem interagir entre si e com humanos;

- Comércio eletrônico – os agentes podem representar seus usuários em busca de realização de negócios.

Para vários autores, a evolução do comércio eletrônico caminha exatamente para o comércio eletrônico baseado em agentes, com agentes representando

empresas em busca de negócios, interagindo para comprar ou vender bens e serviços em ambientes virtuais de negociação. Uma solução completamente adequada para o problema do estágio atual das pesquisas sobre comércio eletrônico: negócios realizados de forma automatizada, nos quais nem a tecnologia nem a quantidade de parceiros se constituam em obstáculos.

2.3 Engenharia de Software Baseada em Agentes

A engenharia de *software* baseada em agentes (AOSE – *Engineering Systems of Software Agents*) ainda é matéria de intensos estudos. Diversas metodologias têm sido desenvolvidas para estruturar uma ferramenta que sistematize as técnicas necessárias para o desenvolvimento adequado de sistemas multiagentes. Embora ainda não exista um padrão para esse tipo de desenvolvimento, a maioria das propostas aborda as fases de análise e projeto de sistemas multiagentes e estende conceitos das técnicas usadas para o desenvolvimento orientado a objeto, como a *Unified Modeling Language* (UML) (BOOCH et al., 2000), por exemplo.

De modo geral, o objetivo da fase de análise é definir modelos dos objetivos, papéis, atividades e interações dos agentes participantes de uma sociedade, evidenciando, desta forma, o que o sistema deve fazer. Já a fase de projeto visa estabelecer uma solução ao problema especificado na fase anterior.

Existem duas abordagens para o desenvolvimento de SMA. A primeira adapta as metodologias existentes para o desenvolvimento orientado objeto, para a fase de análise e projeto. A *Agent Unified Modeling Language* (AUML) (ODELL et al., 2000), por exemplo, estende a UML com notações próprias para o conceito de agentes. O principal problema dessa abordagem é que muitas das características dos agentes são difíceis de modelar apenas com os conceitos da orientação a objeto.

A segunda abordagem se baseia na teoria dos agentes, cumprindo a fase de análise e projeto através da orientação a agentes. Contudo, são, em sua maioria,

incompletas, não existindo um consenso entre elas, cada qual possuindo o seu próprio conjunto de conceitos para o desenvolvimento dos sistemas. É o caso das metodologias Gaia (WOOLDRIDGE et al., 2000) e MAS-CommonKADS (IGLESIAS et al., 1997), que possuem modelos que cobrem as fases de análise e projeto, mas demonstram ter algumas limitações no que diz respeito à representação dos conceitos necessários a orientação a agentes.

Para o desenvolvimento deste trabalho, optamos por utilizar a metodologia MESSAGE (*Methodology for Engineering Systems of Software Agents*) (EVANS et al., 2001), que busca combinar as melhores características das duas metodologias acima, na fase de análise. E, em seguida, utilizamos a AUML, na fase de projeto, para modelar os agentes e suas interações durante o processo de negociação do ICS. Tendo isso em vista, abordaremos essas duas tecnologias a seguir.

2.3.1 MESSAGE

Inicialmente desenvolvida para atender as necessidades da indústria de telecomunicações, a metodologia MESSAGE cobre a maioria dos aspectos fundamentais do desenvolvimento de sistemas multiagentes. Contudo, ela pode ser vista como uma metodologia genérica aplicável também a outros domínios.

De maneira geral, podemos definir a MESSAGE como uma metodologia de engenharia de *software* orientada a agentes, que procura estender as atuais técnicas de desenvolvimento de *software* orientadas a objetos para simplificar e formalizar o processo de desenvolvimento de sistemas multiagentes.

A orientação a agentes utiliza os conceitos de organização e sociedades para descrever como os agentes atuam em conjunto nos SMA, para atingir seus objetivos coletivos. Além disso, combina-os com os conceitos de inteligência artificial e psicologia cognitiva, para descrever os agentes através de idéias e estruturas de um nível mais alto de conceituação, o qual, em inteligência artificial, é conhecido como nível de conhecimento (*knowledge level*).

A MESSAGE usa a UML para modelar os conceitos em nível de dados (*data level*) e adiciona conceitos em nível de conhecimento (como os conceitos de entidades e relacionamentos que são definidos no MESSAGE metamodelo). Alguns desses conceitos em nível de conhecimento utilizados pela MESSAGE são:

- **Agente:** entidade atômica e autônoma capaz de executar alguma função potencialmente útil. Possui serviços, correspondentes às operações na orientação a objeto, e propósitos, que influenciam a maneira como os agentes atendem um pedido de execução de serviço.

- **Organização:** grupo de agentes que trabalham juntos em torno de um propósito comum. Seus serviços são providos e seus propósitos são alcançados coletivamente através dos seus agentes participantes. Possui uma estrutura expressa através do poder dos relacionamentos dos seus constituintes e um comportamento expresso através de suas interações.

- **Papel:** descreve características externas de um agente num contexto particular. Um agente é capaz de exercer diversos papéis e diversos agentes são capazes de exercer um mesmo papel.

- **Recurso:** é utilizado para representar entidades não-autônomas, tais como bancos de dados ou programas externos, que são utilizadas pelos agentes no propósito de alcançar seus objetivos.

- **Tarefa:** unidade de atividade exercida por um único agente. Uma ou mais tarefas são executadas no sentido de o agente atingir seus objetivos.

- **Interação ou Protocolo de Interação:** uma interação possui por definição mais de um agente participante e um propósito que seus participantes coletivamente desejam alcançar. Um protocolo de interações define um padrão de trocas de mensagens associado a uma interação.

- **Objetivo:** é o intuito de um agente de alcançar algum estado desejado. Os objetivos podem ser do tipo geral ou específicos e implicam a execução de tarefas.

A MESSAGE define um conjunto de visões ou perspectivas que permitem uma clara compreensão de diferentes aspectos da modelagem. Cada visão é focaliza apenas um aspecto, mas o conjunto delas fornece um panorama geral do sistema a ser desenvolvido. Algumas dessas visões são descritas a seguir:

- **Visão Organizacional:** mostra os agentes, as organizações, os papéis e os recursos, seus relacionamentos através do diagrama de relacionamentos estruturais e o de familiaridade.
- **Visão de Tarefas e Objetivos:** mostra os objetivos, as tarefas e as dependências entre eles através dos diagramas de objetivos e o de tarefas.
- **Visão de Agentes e Papéis:** mostra os agentes individualmente com seus papéis, objetivos, responsabilidades e recursos por ele utilizados.
- **Visão das Interações:** mostra a forma na qual os agentes ou os papéis trocam informações entre si e com o seu ambiente. O diagrama de interações consiste de um conjunto de interações e de uma representação detalhada em termos de protocolos de interação.
- **Visão do Domínio:** mostra em um domínio, os conceitos e os relacionamentos específicos que são relevantes para o sistema sob desenvolvimento. A MESSAGE utiliza o diagrama de classes da UML para este propósito.

O processo de análise na MESSAGE tem por objetivo produzir um modelo (ou um conjunto de visões) do sistema a ser desenvolvido, bem como seu ambiente. E isso é alcançado através de um refinamento feito em etapas. O nível 0 corresponde ao início desse refinamento e diz respeito à definição do sistema em relação a seus *stakeholders* e ao ambiente no qual o sistema se insere. O sistema é visto como um conjunto de organizações que interagem com seus recursos, atores ou outras organizações. Subseqüentemente, outras etapas de refinamento são modeladas: nível 1, nível 2, e assim por diante, havendo necessidade para tanto. Nesses níveis,

são considerados outros aspectos do sistema, como desempenho, distribuição, tolerância a falhas, segurança, etc. (CAIRE et al., 2001).

Já na fase de projeto, são produzidas entidades computacionais que representem o SMA, que surgiram na fase de análise. Em geral, os artefatos produzidos em cada modelo de análise deverão ser transformados nessas entidades, que, em seguida, serão finalmente implementadas. Para se obterem essas entidades, seguem-se os seguintes passos:

- Refinamento dos artefatos de análise em entidades computacionais: isso envolve, dentre outras coisas, a geração e os refinamentos de casos de usos e diagramas de seqüência da UML;
- Seleção de uma arquitetura para os agentes e o posterior preenchimento dessa arquitetura: arquiteturas cognitivas são requeridas para agentes que possuem necessidades de comunicação e processamento envolvendo raciocínio e aprendizagem. Arquiteturas reativas são usadas em agentes envolvidos em simples mecanismos de controle;
- Atualização e modificação das visões existentes: as decisões necessárias ao refinamento durante a fase de projeto podem afetar as visões existentes, sendo então necessárias sua atualização e modificação;
- Estruturação dos resultados de acordo com a visão organizacional: a visão organizacional é importante no sentido de obter uma representação arquitetural do SMA como um todo.

Como visto, a MESSAGE é uma metodologia para as fases de análise e projeto de SMA e utiliza a UML para modelar o sistema em nível de dados (*data level*) adicionando uma camada de mais alto nível (*knowledge level*) para atender a necessidade de modelagem dos agentes e de suas interações.

2.3.2 AUML

É uma linguagem de modelagem que estende a UML para modelar aspectos relacionados ao paradigma de desenvolvimento orientado a agentes. De modo geral, os diagramas propostos pela UML são centrados nos estados, enquanto na AUML o centro dos diagramas são os agentes ou o processo, quando do programa em execução.

A proposta inicial da AUML, feita pela *Foundation for Intelligent Physical Agents* (FIPA) e pelo grupo OMG *Agent Work* (*Object Management Group*), é representar os protocolos de interação de agentes, os quais descrevem um padrão de comunicação entre eles através de uma seqüência de mensagens e suas restrições de conteúdo. Isso ocorre em três fases ou camadas próximas relacionadas (ODELL et al., 2000):

- **Nível 1** – protocolo geral (modelos e pacotes): onde o protocolo como um todo é tratado como uma entidade independente. Um pacote é uma agregação conceitual de seqüências de interação e um modelo identifica entidades que não se limitam ao pacote, mas que precisam restringidas quando o modelo do pacote estiver sendo instanciado;
- **Nível 2** – interações entre agentes (diagramas de seqüência e colaboração): fornece uma visão alternativa das interações entre os agentes;
- **Nível 3** – processamento interno do agente (diagramas de estados e atividades): detalham o comportamento dos agentes quando estes estiverem executando os protocolos.

Na proposição original, havia um interesse apenas nas camadas dos protocolos de interações, mas atualmente tem crescido o escopo da notação para uma visão mais geral do sistema. Alguns aspectos de interesse do grupo de desenvolvimento da linguagem são:

- **Prover uma visão total do sistema:** a AUML busca representar uma visão geral dos agentes individualmente dentro do sistema e suas relações semânticas;

- **Representar as interações entre os agentes:** a AUML estende diagramas de seqüência a fim de torná-los mais expressivos, permitindo uma notação mais rica para os agentes e representando o paralelismo e a comunicação entre eles;

- **Representar o processamento interno dos agentes:** a AUML usa os diagramas de atividade e estado para prover um maior entendimento do comportamento interno dos agentes.

Os principais diagramas da AUML são:

- **Diagrama de Seqüência** – representa as mensagens trocadas entre agentes, cuja disposição dos elementos gráficos enfatiza a ordem cronológica de comunicações;

- **Diagrama de Colaboração** – enfatiza as associações entre agentes, nas quais a seqüência das interações é representada através da numeração das mensagens, e os agentes podem ser dispostos em qualquer lugar do diagrama;

- **Diagrama de Atividades** – representa operações e os eventos que as ativam estas, diferindo dos diagramas de interação por representar de maneira explícita as linhas de execução dos fluxos de controle, o que é particularmente útil para protocolos de interação complexos envolvendo paralelismo de processamento; e

- **Diagrama de Estado** – tem uma visão centrada nos estados, sendo melhor aplicável como um mecanismo de restrições para os protocolos de interação.

O trabalho desenvolvido na definição da AUML constitui um esforço no sentido de estender os diagramas da linguagem UML às características inerentes ao paradigma orientado a agentes. Porém, por se tratar de uma proposta bastante recente, é natural que ainda não exista consenso sobre a sua real adequação à modelagem de um SMA. Atualmente, estão sendo estudados quais os diagramas da UML que podem se estender à AUML, a fim de atender todas as características e particularidades dos agentes.

3 ICS – INTELLIGENT COMMERCE SYSTEM

Neste capítulo, apresentamos uma descrição do ambiente ICS de suporte ao comércio eletrônico B2B (*business-to-business*). Descrevemos o seu ciclo de vida do comércio eletrônico proposto em suas cinco fases e sua arquitetura através da descrição de seus componentes.

3.1 Introdução

Embora atualmente o comércio eletrônico esteja vivenciando um crescimento acelerado, já atingindo cifras significativas, observa-se que ainda existem desafios tecnológicos, de estruturação e de processos que necessitam ser vencidos. Apesar disso, o comércio eletrônico está abrindo novas fronteiras, possibilitando às empresas a formação de novos mercados e de negócios de maior valor agregado, devido principalmente ao caráter global de sua abrangência e à redução dos custos para implementação de produtos ou serviços que propiciem grande vantagem competitiva.

O ICS está sendo desenvolvido com o objetivo de proporcionar às empresas um ambiente automatizado de comércio eletrônico B2B, através do uso de agentes inteligentes móveis e estacionários, facilitando a localização de parceiros comerciais, a negociação entre eles e o fechamento de contratos.

Dentro do sistema, as empresas são representadas pelos agentes móveis que negociam a compra e a venda de produtos ou serviços, enquanto os agentes estacionários são responsáveis por funções internas que possibilitam a troca de informações necessárias à negociação e ao fechamento de contratos. Uma contribuição importante do sistema é a utilização de ontologias e repositórios de dados para prover conhecimento de forma compartilhada aos agentes. Todo o processo do comércio eletrônico dentro do ICS é descrito através do seu ciclo de vida, desde a modelagem do usuário, passando pela negociação, chegando até a

formação e o cumprimento de contratos firmados pelas empresas representadas no sistema.

Nas seções seguintes, abordaremos essas características.

3.2 Ciclo de Vida do Comércio Eletrônico no ICS

Estendendo o ciclo de vida proposto por Jennings et al. (1996) e Bartolini (2003), o ICS inclui a fase de modelagem do usuário, o conceito de *feedback* de informações e o emprego de ontologias para prover uma base de conhecimento compartilhada entre os agentes (LABIDI; FONSECA, 2003). A Figura 2 apresenta o ciclo de vida do comércio eletrônico proposto pelo ICS em suas cinco fases.

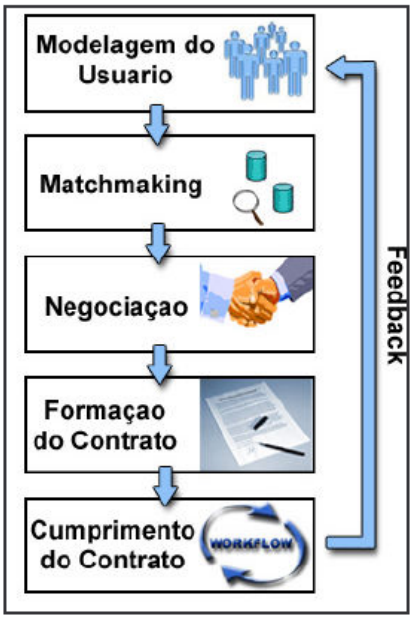

Figura 2: Ciclo de vida do ICS

3.2.1 Modelagem do usuário

Fase na qual são adquiridos os perfis dos usuários (empresas). Seus comportamentos, preferências, características, restrições, tipos de negociação de que deseja participar (leilão, compra direta, concorrências, etc.), além das informações específicas dos produtos ou serviços que desejam negociar ou

contratar. Com base nessas informações, os agentes negociantes, que efetivamente representam os usuários do sistema, formarão suas estratégias de negociação, bem como seus anúncios, que ficarão disponíveis num repositório de propagandas compartilhado (BASTOS FILHO, 2003).

3.2.2 *Matchmaking*

Processo no qual, a partir das informações capturadas na fase de modelagem do usuário, os agentes negociantes que possuem interesses afins são colocados em contato com potenciais parceiros de negócios. O agente *matchmaker* é o responsável por essa fase, possuindo o objetivo de executar o *matching* entres os agentes negociantes que representam as empresas. Isso é alcançado através da consulta do repositório de anúncios e descobrindo oportunidades de negócios dentre aqueles anunciados (TOMAZ, 2003). Ao final dessa fase, o agente *matchmaker* deverá formar um *cluster* de agentes negociantes que desejam comprar ou vender o mesmo produto, ou ainda contratar o mesmo tipo de serviço.

3.2.3 Negociação

Os agentes negociantes reunidos iniciam o processo de negociação para efetivar suas transações e alcançar seus objetivos. Um agente mediador é designado para agir como árbitro da negociação, controlando o fluxo de comunicações e estabelecendo os procedimentos necessários para o sucesso desse processo. O agente mediador deve garantir a “justiça” entre os negociantes, utilizando o protocolo de negociação (FONSECA, 2003). Maiores detalhes a respeito dessa fase serão apresentados no capítulo 4 deste trabalho, através do estudo do modelo de negociação.

3.2.4 Formação do contrato

Após a negociação, os agentes negociantes devem formalizar os acordos alcançados através de contratos eletrônicos com validade jurídica. Esses contratos, que devem conter os termos acordados pelas empresas (como preço, prazo, forma

de pagamento, entrega, etc.), são assinados eletronicamente pelas partes, utilizando-se certificadoras digitais como forma de assegurar sua autenticidade, integridade e legalidade diante das normas jurídicas estabelecidas. Um agente de contrato fica responsável por essa fase.

3.2.5 Cumprimento do contrato

Consiste em acompanhar o cumprimento das obrigações assumidas pelas partes, através dos contratos assinados na fase anterior. As empresas envolvidas se mantêm informadas pelo sistema da atuação dos seus parceiros comerciais no que tange ao cumprimento do contrato firmado, possibilitando assim tomadas de decisão corretivas (quebra de contrato), caso seja necessário. Por outro lado, esse acompanhamento pode enriquecer o modelo de usuário na fase de sua modelagem, através da atualização do seu comportamento numa dada negociação, o que caracteriza o conceito de *feedback* no ICS (OLIVEIRA, 2004).

Um contrato pode ser considerado como um processo de negócio cujas obrigações funcionam como atividades que devem ser cumpridas dentro de prazos previamente estabelecidos. Desse modo, tecnologias de *workflow* temporal (LABIDI et al., 2003) são utilizadas para controlar o fluxo de execução dos contratos firmados dentro do ICS.

3.3 Arquitetura do ICS

O sistema ICS foi concebido para proporcionar uma arquitetura aberta, flexível e evolutiva, utilizando-se um ambiente aberto, como a internet, para possibilitar a entrada e a saída, no sistema, dos agentes móveis que representam as empresas negociantes. Dessa forma, os agentes negociantes podem, sempre que desejado, migrar através da rede para um lugar comum (*marketplace*) e iniciar o processo de negociação.

A arquitetura do ICS, que é baseada no ciclo de vida exposto acima, é mostrada na Figura 3, sendo composta pelos seguintes componentes: *marketplace*, região, agente *matchmaker*, agentes negociantes (compradores e vendedores), agente mediador, agente de contrato, repositórios de ontologia, de propagandas, de estereótipos, de *log*, de protocolos de negociação e de contratos.

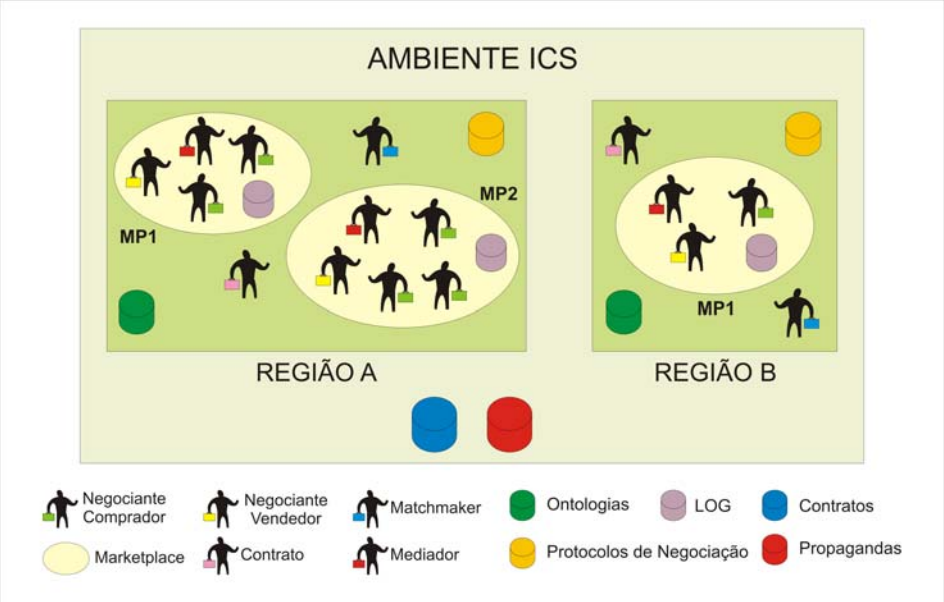

Figura 3: Arquitetura do ICS

Marketplace: espaço virtual instanciado numa máquina (*host*) cujos agentes podem desempenhar suas funções específicas. Os agentes se movem pela rede viajando entre *marketplaces* em busca de cumprir seus objetivos. O *marketplace* provê uma estrutura de comunicação, ontologias, propagandas, identificação e mediação para que os agentes possam negociar seus produtos e fechar contratos.

Região: conjunto de *marketplaces* que se utiliza de uma mesma ontologia de domínio, isto é, operando dentro do mesmo tipo de negócio (vestuário, matérias-primas, alimentação, etc.). Isso facilita a comunicação entre os agentes, pois providencia um vocabulário comum a ser usado na negociação. Cada *marketplace* é identificado com apenas uma região, podendo haver mais de um *marketplace* dentro da mesma região.

Repositório de ontologias: base de dados onde ficam armazenadas as ontologias de domínio e de negociação. Cada ontologia armazenada nesse repositório está associada a um tipo de negócio e a uma região específica. Para o sistema trabalhar com um novo tipo de negócio, basta adicionar as ontologias respectivas no repositório.

Repositório de propagandas: base de dados onde ficam armazenados os anúncios dos agentes negociantes. Funciona similarmente a uma lista de “páginas amarelas”, onde cada agente pode declarar o que quer negociar e sob quais condições. É utilizado pelo agente *matchmaker* para encontrar potenciais parceiros de negócios.

Repositório de estereótipos: base de dados onde se armazena o perfil de cada usuário. Utilizado pelos agentes negociantes para descrever as preferências das empresas que são por eles representadas.

Repositório de log: base de dados cujo agente mediador armazena todas as comunicações realizadas entre os agentes negociantes durante a negociação. Utilizado em conjunto com o protocolo de negociação para verificar a possibilidade de acordos possíveis, além de servir para fins de auditoria.

Repositório de protocolos de negociação: base de dados onde ficam armazenados os protocolos de negociação sob forma de ontologias. O protocolo de negociação é um conjunto regras (passos) que deve ser seguido dentro da negociação. Esse repositório é utilizado pelo agente mediador e pelos negociantes.

Repositório de contratos: base de dados onde ficam armazenados os contratos de negociações bem sucedidas, com o propósito de registrar os negócios realizados no ICS entre as empresas e facilitar a criação de novos contratos. Esse repositório é acessado apenas pelo agente de contrato.

Agente *matchmaker*: é o responsável por localizar potenciais parceiros de negócios. Utiliza-se do repositório de propagandas para atingir seus objetivos. Ao localizar um conjunto de agentes negociantes afins (*cluster* de negociantes), deve

instanciar um *marketplace* e um agente mediador que controlará a negociação dentro do *marketplace* respectivo.

Agentes negociantes: são instanciados pelas empresas participantes do sistema e as representam. Podem ser compradores ou vendedores de produtos ou serviços. Carregam em si os interesses das empresas através da captura das informações adquiridas durante o processo de modelagem do usuário.

Agente mediador: é instanciado pelo agente *matchmaker* e é responsável pela negociação dentro de um *marketplace* específico. Através do uso do protocolo de negociação, controla o fluxo de comunicações entre os agentes negociantes e assegura a imparcialidade da negociação. Faz o registro de todas as comunicações no repositório de *log* e, ao término da negociação, em caso de acordo entre as partes, instancia o agente de contrato para formalização do contrato, exercendo assim, um papel fundamental durante todo o processo de negociação. Maiores detalhes sobre esse agente serão fornecidos no capítulo 5 deste trabalho.

Agente de contrato: responsável pelo gerenciamento da constituição dos contratos dentro do ICS. Exerce sua função, dentre outras coisas, comunicando-se com agentes negociantes que chegaram a um acordo na negociação, acessando o repositório de contratos para recuperar contratos firmados previamente ou registrar novos contratos, validando-os mediante consulta ao protocolo de negociação.

3.4 Conclusão

Como visto, o ICS apresenta avanços no que tange ao desenvolvimento de sistemas para comércio eletrônico B2B. O emprego da tecnologia de agentes inteligentes móveis possibilita um processo de negociação mais eficiente, pois torna o ambiente mais flexível, facilitando a localização e o agrupamento de potenciais parceiros de negócios.

A arquitetura baseada num ciclo de vida bem definido possibilita o desenvolvimento do sistema através de seus componentes, resultando numa maior qualidade e robustez do produto final.

Por usar ontologias de acesso compartilhado pelos agentes para lhes prover conhecimento, o ICS permite que esses possam negociar sobre qualquer domínio de negócio (livros, vestuários, alimentação, etc.) e utilizando qualquer tipo de negociação (leilão, compra direta, concorrências, etc.).

Abordaremos, no capítulo 4, a seguir, o modelo de negociação que servirá como subsídio para o desenvolvimento do agente mediador.

4 MODELO DE NEGOCIAÇÃO NO ICS

Neste capítulo, iniciamos apresentando um estudo de alguns trabalhos relacionados que tratam da negociação entre agentes. Em seguida, apresentamos o modelo de negociação do ambiente ICS, resultante do refinamento do modelo apresentado por Fonseca (2003).

4.1 Introdução

Para um melhor entendimento de como uma negociação deve transcorrer entre os agentes, propomos o modelo da negociação do ambiente ICS que é resultante do refinamento do modelo inicialmente delineado em Fonseca (2003). Esse refinamento se dá através da construção dos diagramas gerados na metodologia MESSAGE, que permitem obter uma melhor compreensão de como os agentes se organizam e da contextualização do modelo dentro do ambiente.

Ao detalharmos o modelo, pudemos esclarecer pontos que estavam apenas implícitos na proposta inicial, e assim, obtivemos uma visão mais clara do processo de negociação como um todo. Isso proporcionou condições para que fossem feitas análises mais precisas dos objetivos e tarefas do sistema, além de contribuir para um melhor entendimento do papel do agente mediador que é tratado no capítulo 5.

Iniciamos nossa abordagem com um estudo de trabalhos relacionados que tratam da negociação entre agentes inteligentes.

4.2 Trabalhos Relacionados

Na área de comércio eletrônico, existem alguns trabalhos relacionados à negociação entre agentes inteligentes que representam seus usuários. Na sua grande maioria, esses sistemas trabalham com apenas um tipo de negociação, pois

normalmente codificam o protocolo de negociação internamente nos agentes que estão em negociação. É o caso dos seguintes ambientes:

- **Kasbah** – do MIT *Media Laboratory* (*Massachusetts Institute of Technology*), implementa o *continuous double auction*, um tipo de leilão amplamente utilizado em bolsas de valores eletrônicas. Os agentes participantes estão predefinidos no sistema e utilizam um protocolo interno próprio, não sendo permitido aos usuários a criação de agentes com suas características particulares (CHAVEZ; MAES, 1996);

- **SMACE** – Sistema Multi-Agente para Comércio Eletrônico (CARDOSO, 1999), implementa o leilão inglês. Não possui um agente que atue como gerenciador da negociação, sendo necessário que o protocolo de negociação esteja codificado internamente nos participantes;

- **CAAT** – *Collaborative Agreement for Automatic Trading* (NCHO; AIMEUR, 2004), implementa o leilão inglês. Possui o protocolo de interações implementado ou nos seus participantes (quando a negociação ocorre sem mediação), ou no agente conhecido como *trade facilitator*.

Além desses, outros trabalhos foram pesquisados. Dentre eles, o que mais se aproxima da abordagem proposta pelo ICS é o que está sendo desenvolvido pelo *Hewllet-Packard Laboratories* (HP Lab), descrito em Bartolini et al. (2003). Essa equipe vem desenvolvendo um *framework* de interações genérico para negociações automatizadas, através da definição de um protocolo de negociação genérico que é utilizado pelo *framework*. Esse *framework* pode ser parametrizado com diferentes regras de negociação e, dependendo da escolha do conjunto de regras, diferentes mecanismos de negociação podem ser implementados.

O trabalho do HP Lab possui alguns pontos em comum com o ICS, dos quais podemos destacar como mais relevantes:

- O protocolo de negociação não é codificado internamente nos agentes participantes da negociação, o que possibilita uma maior flexibilidade nas possíveis formas de negociação;

- A negociação é feita entre *agentes participantes* (similares aos negociantes) com a participação do *negotiation host* (similar ao mediador). Os participantes podem fazer propostas e acordar sobre quais regras negociar, enquanto o *host* é responsável pela imposição das regras e do protocolo que governam a negociação.

Apesar de possuírem pontos em comum, os trabalhos diferem em alguns aspectos importantes, tais como os citados na Tabela 2.

Tabela 2: Diferentes aspectos entre o ICS e o trabalho do HP Lab.

HP Lab	Ambiente ICS
O protocolo de negociação é simplificado e genérico, isto é, pode ser utilizado para qualquer tipo de negociação. Um conjunto de regras serve de parâmetro para o *framework*, e determina o tipo de negociação que irá ocorrer entre os agentes.	Para cada tipo de negociação, existe um protocolo específico que determina as regras que conduzirão a negociação entre os agentes.
O *negotiation host* não é apenas um agente, mas na verdade, um sistema multiagentes que possui agentes para diversos papéis dentro da negociação. Um agente participante pode atuar como *host,* bastando, para isso, criar os seguintes agentes subordinados: ▪ *Gatekeeper* – responsável pela admissão dos agentes no ambiente de negociação; ▪ *Proposal Validator* – assegura que as propostas estejam corretamente elaboradas no que diz respeito a sua forma; ▪ *Protocol Enforcer* – assegura que as propostas estejam de acordo com as regras de negociação. ▪ *Agreement Maker* – assegura que os acordos sejam formados seguindo as regras de negociação; ▪ *Information Updater* – notifica os participantes do atual estado da negociação (quantos participam, quem vende, quem compra, etc.); ▪ *Negotiation Terminator* – declara o fim da negociação de acordo com as regras que a estabelecem.	O papel exercido pelo *negotiation host* e seus subordinados fica a cargo apenas dos seguintes agentes: ▪ *Matchmaker* – responsável pela formação do *cluster* de negociação, propiciando, assim, a admissão dos negociantes no *marketplace* por ele instanciado (*gatekeeper*); ▪ de contrato – responsável pela formalização dos acordos alcançados, seguindo o protocolo de negociação (*agreement maker*); ▪ Mediador – responsável pelas atividades exercidas pelos seguintes agentes: *proposal validator*; *protocol enforcer*; *information updater*; e *negotiation terminator*.
Um agente atuando como *host* pode também atuar como participante (na negociação um-para-um, por exemplo) ou não participar diretamente da negociação (no caso de atuar como leiloeiro em leilões). Em outros casos, o papel de *negotiation host* pode passar de um agente para outro com a negociação em andamento.	O agente mediador é especializado no gerenciamento da negociação, pertencendo a ele, e somente a ele, a tarefa de coordenar a negociação dentro do *marketplace*.

Como podemos observar, a solução proposta pelo ICS de um modo geral, torna-se melhor por optar pela simplicidade. O trabalho de implementação é menos

complexo, além de proporcionar uma redução no número de trocas de mensagens entre os agentes e, dessa forma, tornando o sistema menos propenso a erros, o que pode acontecer, caso tenhamos um grande número de agentes negociantes realizando a negociação.

Outro aspecto relevante diz respeito ao papel exercido pelos agentes participantes, que, em determinadas ocasiões, podem exercer o papel de *host*. Isso faz com que haja um enfraquecimento na credibilidade do sistema, pois é atribuído ao usuário a tarefa de gerenciar a negociação, podendo ocasionar desconfianças dos demais parceiros de negócio em relação à imparcialidade do *host*.

No ICS, o agente mediador é proposto como uma entidade independente (isenta) aos usuários. Uma vez que estes confiem no sistema como um todo, concordando com as condições impostas pelo próprio sistema, não há por que duvidar da imparcialidade do mediador como gerenciador da negociação.

4.3 Processo de Negociação

A negociação é fase onde os agentes negociantes buscam realizar negócios com os demais agentes parceiros, todos pertencentes a um mesmo *cluster* de negociantes previamente identificado pelo agente *matchmaker*. Um agente mediador é instanciado para ficar responsável pela negociação: controlando o fluxo de mensagens entre os negociantes através do uso do protocolo de negociação e mantendo um registro de todas as comunicações realizadas entre os agentes, para fins de auditoria. O processo de negociação no ICS é simplificado para os passos descritos a seguir.

4.3.1 Pré-condições

Para que se possa iniciar a negociação são necessárias as seguintes pré-condições, que ficam sob a responsabilidade do agente *matchmaker*:

- **Formação de um *cluster* de agentes negociantes**: atividade resultante da fase de *matchmaking*, onde são localizados agentes negociantes que atuam sob um mesmo domínio e possuem interesses complementares, tornando-se potenciais parceiros de negócios.

- **Criação de um *marketplace***: ambiente virtual no qual os agentes negociantes irão interagir em busca da realização de suas necessidades.

- **Instanciação de um agente mediador**: agente que ficará responsável pela negociação, garantindo sua imparcialidade e assegurando que o protocolo de negociação seja cumprido.

4.3.2 Cenário básico

Entenda-se por cenário básico aquele no qual os passos seguidos pelos agentes participantes os leva a uma negociação bem sucedida, tendo ao final desta, um acordo estabelecido ou não. Não é coberto por este cenário erros de comunicação entre os agentes ou desvios de procedimentos. Este cenário básico, pode ainda variar de acordo com o tipo de negociação a ser utilizado.

- **Passo 1**: De posse do *cluster* de negociantes, o agente mediador deve convocar os mesmos para o *marketplace* onde ocorrerá o processo de negociação.

- **Passo 2**: Os agentes negociantes por sua vez devem atualizar suas propagandas para indicar que já estão em negociação e desta forma evitar que agente *matchmaker* os considere em um novo processo de matchmaking, evitando assim convocações simultâneas.

- **Passo 3**: Em seguida os negociantes deslocam-se para o local designado pelo mediador e o informam que já estão prontos para iniciar o processo de negociação.

- **Passo 4**: O agente mediador deve informar aos participantes quantos agentes estão envolvidos na negociação e qual protocolo de negociação será

utilizado. Isto é importante, pois os agentes necessitam destas informações para saber se comportar de acordo com o protocolo e para que possam traçar suas estratégias de negociação.

- **Passo 5**: O agente mediador solicita então as propostas de negócio para os agentes determinados pelo protocolo de negociação estabelecido. Por exemplo, em caso de leilão direto, solicita as propostas dos compradores; em caso de leilão reverso, as propostas são solicitadas aos vendedores.

- **Passo 6**: Os agentes negociantes elaboram suas propostas de acordo com o protocolo de negociação e suas próprias estratégias e em seguida as enviam ao agente mediador.

- **Passo 7**: O agente mediador ao receber as propostas dos negociantes verifica se as propostas estão de acordo com o protocolo de negociação e as repassa para as contrapartes que são determinadas pelo protocolo.

- **Passo 8**: As contrapartes ao receberem as propostas devem verificar se estas satisfazem suas necessidades, de acordo com o estabelecido na modelagem do agente e na sua estratégia interna. Em seguida responder ao mediador se um acordo foi atingido ou não. Um acordo é atingido se a proposta recebida cumpre as necessidades procuradas pelo agente e atende aos critérios definidos pelo usuário.

- **Passo 9**: O mediador recebe a resposta das propostas e verifica 2 alternativas:

  i) **A negociação continua**: situação na qual os agentes ainda não atingiram um acordo, mas ainda existe espaço para continuar a negociação. O mediador inicia uma nova rodada de negociação solicitando novas propostas aos agentes determinados pelo protocolo de negociação.

  ii) **A negociação é encerrada**: a negociação termina quando não há mais acordos possíveis no *cluster*, tendo ou não sido alcançados acordos em rodadas anteriores. O mediador pode chegar a essa conclusão ao receber, em uma rodada de negociação, as mesmas propostas da rodada

anterior. Isso porque, em cada rodada de negociação r, os agentes devem fazer propostas melhores que a melhor proposta da rodada de negociação r-1. Se não é feita nenhuma proposta melhor que a da rodada anterior, isso significa que não há mais negociações possíveis, e o *cluster* deve ser dispensado. Neste momento, o mediador verifica se houve algum acordo anteriormente e quais negociantes estão envolvidos. Em caso positivo, ele instancia um agente de contrato para a formalização do acordo alcançado e avisa os respectivos parceiros.

Durante todo o processo de negociação o agente mediador deve ainda ficar responsável pelas seguintes tarefas:

- **Registro de mensagens**: o mediador deve registrar em um repositório de log todas as comunicações entre os agentes para que ele possa estabelecer a existência de acordos possíveis e também para fins de auditoria.
- **Verificação de desistências**: o mediador deve verificar se durante o processo de negociação houve alguma desistência por parte dos negociantes e comunicar os demais do ocorrido para que eles possam aprimorar suas estratégias.
- **Verificação da regra de finalização**: o mediador deve verificar o protocolo de negociação e o *log* de comunicações para finalizar a negociação caso não seja mais possível um acordo, evitando assim *live locks*, ou seja, os agentes negociantes entrarem em negociação sem chegar a um denominador comum.

4.3.3 Pós-condições

Ao final do processo de negociação, em caso de acordo, o mediador deve instanciar um agente de contrato para gerenciar a formalização do acordo. Em seguida, ele deve dispensar os demais agentes do *cluster* para que eles possam retornar às suas origens e atualizarem suas propagandas, sinalizando que estão prontos para outra negociação.

4.4 Modelagem da Negociação

Iniciando a modelagem com a fase de análise, observamos que pela metodologia MESSAGE, as entidades que compõem o sistema são representadas pelos seus papéis. No caso do ICS, porém, as entidades já estão identificadas e definidas anteriormente como agentes. Sendo assim, para o desenvolvimento do nosso trabalho, definimos primeiramente o contexto no qual o processo de negociação ocorre dentro do ICS, através da Visão Organizacional e da Visão de Objetivos e Tarefas da metodologia MESSAGE.

A metodologia MESSAGE permitiu obter-se um melhor entendimento de como os agentes estão inseridos dentro do sistema e como, em termos gerais, se relacionam e utilizam os recursos à disposição dos mesmos.

Em seguida, na fase de projeto, utilizamos as extensões da AUML, pois assim, foi possível descrever os múltiplos agentes e as interações do mediador com os quais ele estabelece alguma forma de contato.

4.4.1 Visão organizacional

Para compor a visão organizacional, deve-se situar o sistema analisado na estrutura organizacional em que ele será inserido e definir os relacionamentos de familiaridade (*acquaintance relationships*) que suas entidades possuem.

No nosso trabalho, estabelecemos que a negociação é o sistema a ser analisado, o qual, por sua vez, está inserido numa estrutura organizacional maior, que é o próprio ICS como um todo. A negociação é parte integrante do sistema, mas não como uma entidade, e sim como um conjunto de interações entre um agente mediador e agentes negociantes, sendo esses últimos representações de usuários externos (as empresas que desejam negociar). Há ainda a participação do agente *matchmaker* e de um agente de contrato, no caso de algum acordo ser alcançado. E todos eles fazem uso de vários repositórios de dados dentro do sistema, conforme se pode observar na Figura 4.

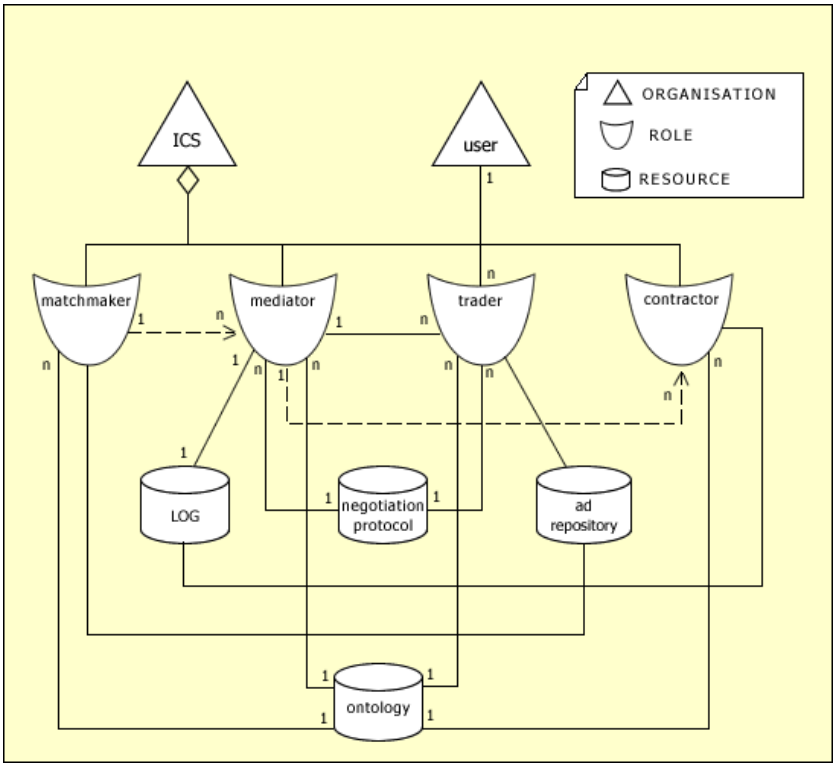

Figura 4: Diagrama organizacional (relacionamentos estruturais)

A partir do diagrama organizacional, podemos identificar as interações existentes entre as entidades no modelo de negociação. O diagrama de familiaridade, mostrado na Figura 5, indica essas interações.

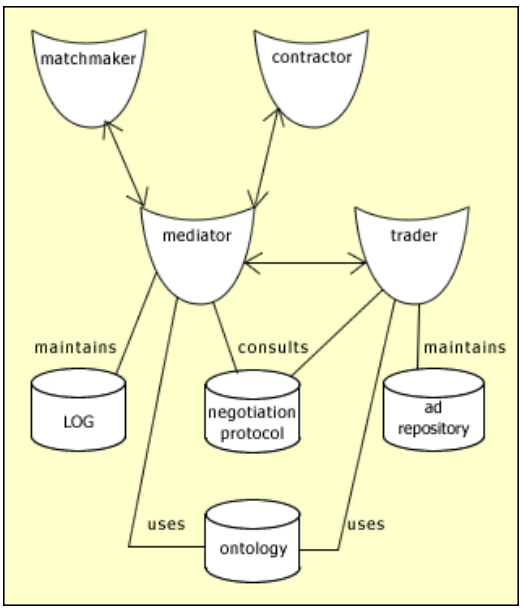

Figura 5: Diagrama de familiaridade

4.4.2 Visão de objetivos e tarefas

A visão de objetivos e tarefas procura mostrar os objetivos do sistema e as tarefas que precisam ser executadas para atingi-los. A partir do processo de negociação apresentado e da visão organizacional proposta, infere-se o diagrama de objetivos, composto por objetivos geral e específicos, conforme mostrado na Figura 6.

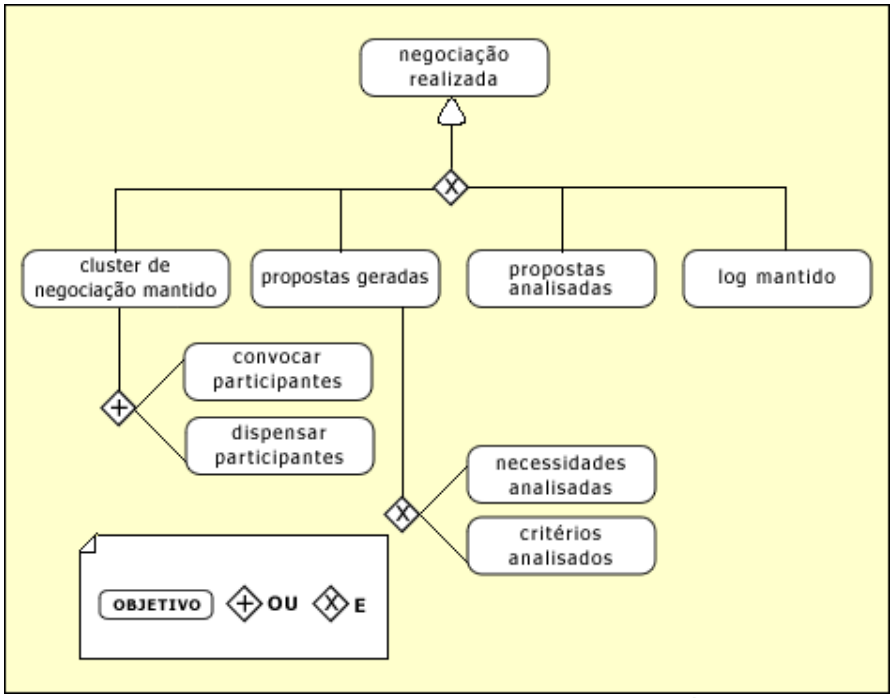

Figura 6: Diagrama de objetivos (geral e específicos)

Abaixo, é feito um detalhamento dos objetivos geral e específicos relacionados à negociação no ambiente ICS, evidenciando-se os papéis, as responsabilidades, as atividades e os agentes relacionados a cada objetivo.

A partir dos objetivos específicos, apresentamos os diagramas de tarefas, contendo as tarefas necessárias para atingi-los. A metodologia MESSAGE faz uso, nesse caso, da notação dos diagramas de atividade da UML. Optamos por também inserir *swim lanes*, identificando quais são os agentes responsáveis por desempenhar essas atividades, quando há mais de um agente envolvido.

Objetivo Geral:

Negociação Realizada

O sistema deve ser capaz de dar suporte à negociação entre agentes negociantes, que representam compradores e vendedores, com a participação de um agente mediador que atua intermediando e gerenciando todo esse processo.

Objetivos Específicos:

Cluster de Negociação Mantido

Papel: gerenciador de *cluster* de negociação

Responsabilidade: gerenciar *cluster* de negociação

Atividades: enviar convocação ou dispensa de participantes; mover-se para local designado; e atualizar o repositório de propagandas.

Agentes: mediador e negociantes

A Figura 7 apresenta o diagrama de tarefas referente ao objetivo acima.

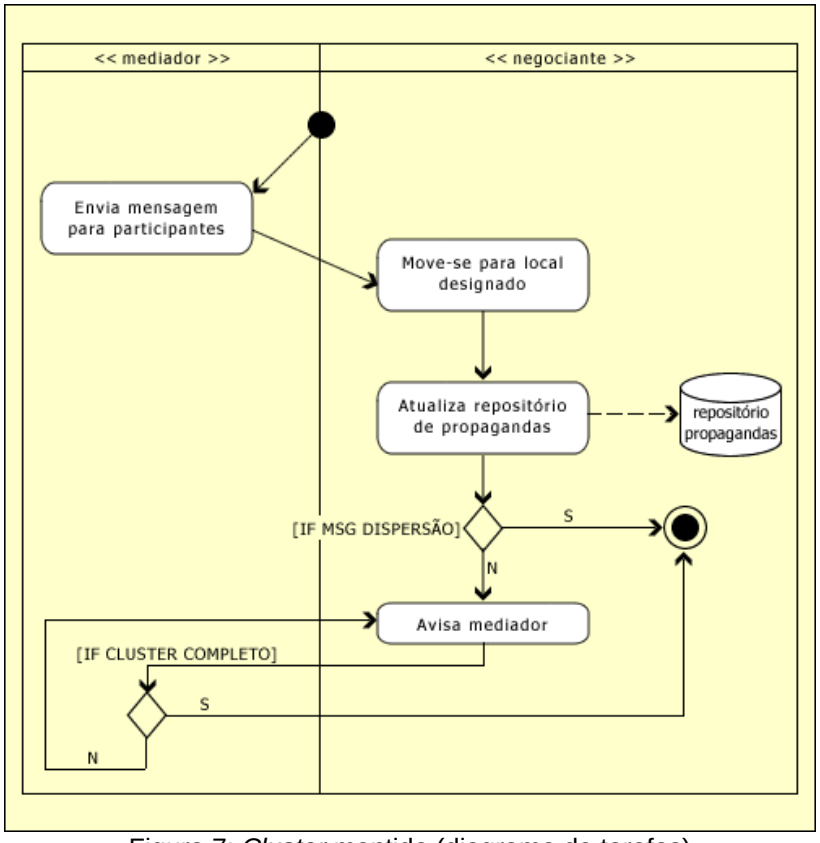

Figura 7: *Cluster* mantido (diagrama de tarefas)

Propostas Geradas

Papel: gerador de propostas

Responsabilidade: gerar propostas

Atividades: receber mensagem sobre proposta (pedido de proposta ou proposta para avaliação); analisar mensagem (verificar atendimento de necessidades e critérios); elaborar resposta de acordo com protocolo de negociação e sua estratégia interna; e enviar resposta.

Recursos: protocolo de negociação e estratégias internas

Agente: negociante

A Figura 8 apresenta o diagrama de tarefas referente ao objetivo acima.

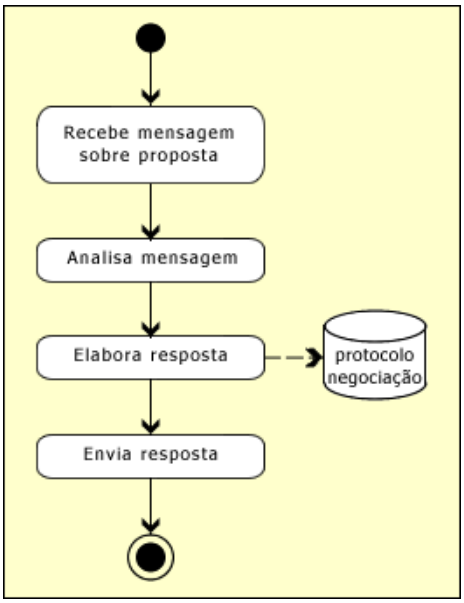

Figura 8: Propostas geradas (diagrama de tarefas)

Propostas Analisadas

Papel: analisador de propostas

Responsabilidade: validar proposta

Atividades: receber proposta; autenticar proposta (verificar correção e validade segundo o protocolo de negociação); e encaminhar proposta.

Recursos: protocolo de negociação e repositório de *log*

Agente: mediador

A figura 9 apresenta o diagrama de tarefas referente ao objetivo acima.

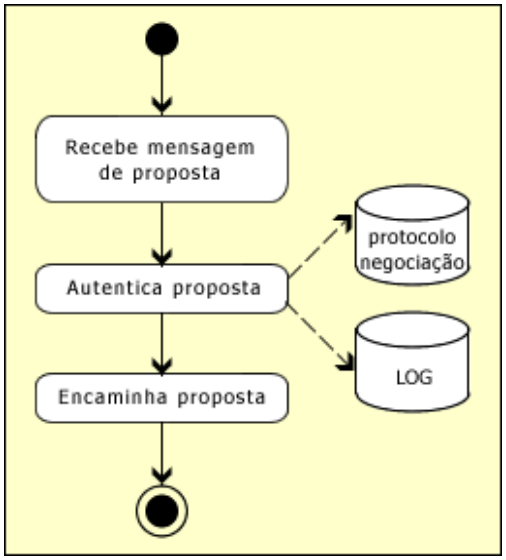

Figura 9: Propostas analisadas (diagrama de tarefas)

Log Mantido

Papel: gerenciador de *log*

Responsabilidade: manter *log*

Atividade: receber e armazenar mensagens no repositório de *log*

Recurso: repositório de *log*

Agente: mediador

A Figura 10 apresenta o diagrama de tarefas referente ao objetivo acima.

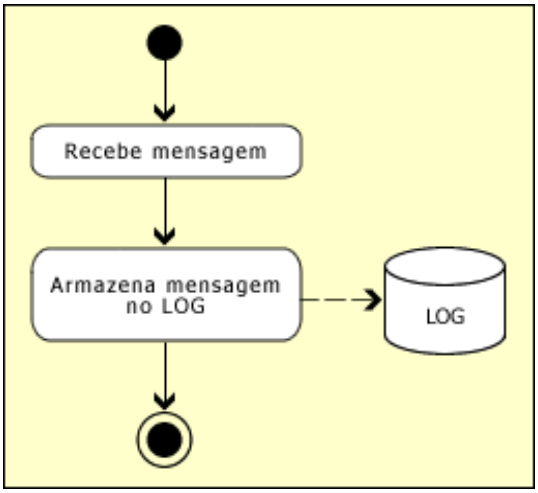

Figura 10: Log mantido (diagrama de tarefas)

4.5 Conclusão

O processo de negociação é a fase onde os agentes negociantes buscam realizar seus objetivos estabelecendo acordos e satisfazendo as necessidades dos usuários do sistema.

Observamos que, apesar de existirem outros trabalhos que abordam a negociação entre agentes que representam seus usuários, o ICS se diferencia pela simplicidade e pelo dos protocolos de negociação externos aos agentes, proporcionando a capacidade de lidar com diversos tipos de negociação.

A modelagem do processo de negociação é uma ferramenta importante para obtenção de uma melhor compreensão desse processo como todo. Além de proporcionar a identificação das tarefas relacionadas aos agentes que estão interagindo em busca dos seus objetivos.

O agente mediador desempenha um papel importante neste processo, impondo o protocolo de negociação, controlando o fluxo de comunicação e garantindo a imparcialidade da negociação. No capítulo 5, iremos abordar a modelagem do agente mediador dando continuidade a este trabalho.

5 AGENTE MEDIADOR

Neste capítulo, apresentamos o processo de modelagem do agente mediador, que é o responsável pela negociação entre os agentes negociantes dentro de um mercado virtual (*marketplace*) anteriormente instanciado pelo agente *matchmaker*.

5.1 Introdução

A fase de negociação do ambiente ICS tem por objetivo buscar acordos entre os agentes negociantes, que buscam satisfazer suas necessidades obedecendo às preferências e restrições impostas pelos usuários do sistema.

Para esse fim, a negociação conta com o suporte do agente mediador como gerenciador do processo, sendo ele o responsável pela coordenação das trocas de mensagens entre os agentes negociantes e pela imposição do protocolo de negociação como forma de assegurar o perfeito andamento da negociação.

Por ser isento, o agente mediador atua de forma independente em relação aos negociantes, proporcionando um gerenciamento imparcial das interações que ocorrem entre esses últimos agentes mencionados. Ao final do processo de negociação, no caso de haver algum acordo estabelecido entre os negociantes, o agente mediador deve instanciar um agente de contrato para formalizar esse acordo na forma de um contrato que tenha valor legal.

Tendo como ponto de partida as características do agente mediador citadas acima, propomos a sua modelagem.

5.2 Funções do Agente Mediador

Como mencionado anteriormente, o agente mediador exerce um papel importante durante todo o processo de negociação, sendo o responsável pela negociação estabelecida dentro de um marketplace específico.

Para atingir seus objetivos, o agente mediador deve assegurar o cumprimento do protocolo de negociação em uso. Todo o fluxo de comunicação entre os agentes deve passar pelo mediador, para que esse possa verificar se as mensagens estão de acordo com o protocolo, tanto em sua composição (forma como a mensagem deve ser composta), quanto na sua ordenação (a seqüência a ser seguida na troca de mensagens entre os agentes), garantindo assim, a imparcialidade em todo o processo.

Além dessas atribuições, o mediador deve também:

- Convocar o *cluster* de negociantes, previamente identificado pelo agente *matchmaker*, para o *marketplace*;
- Avisar aos participantes qual protocolo será usado na negociação;
- Manter os participantes informados sobre a quantidade de agentes envolvidos na negociação, para que esses possam estabelecer suas estratégias;
- Registrar todas as comunicações num repositório de *log*;
- Verificar se a regra de finalização de negociação foi atingida, para evitar que os agentes fiquem presos em *live locks*.
- Dispensar os agentes do *cluster* quando do fim da negociação ou sob solicitação dos mesmos; e
- Instanciar um agente de contrato, caso algum acordo seja alcançado.

5.3 Casos de uso

Tendo em vista os requisitos necessários para o desenvolvimento do agente mediador e o modelo de negociação obtido na fase de análise, iniciamos a fase de projeto. Nessa fase, fazemos uso dos diagramas da UML e as extensões da AUML para modelar o agente mediador e suas interações com os múltiplos agentes em negociação.

Começamos então, com o levantamento dos casos de uso relacionados ao processo de negociação, como demonstrado na Figura 11.

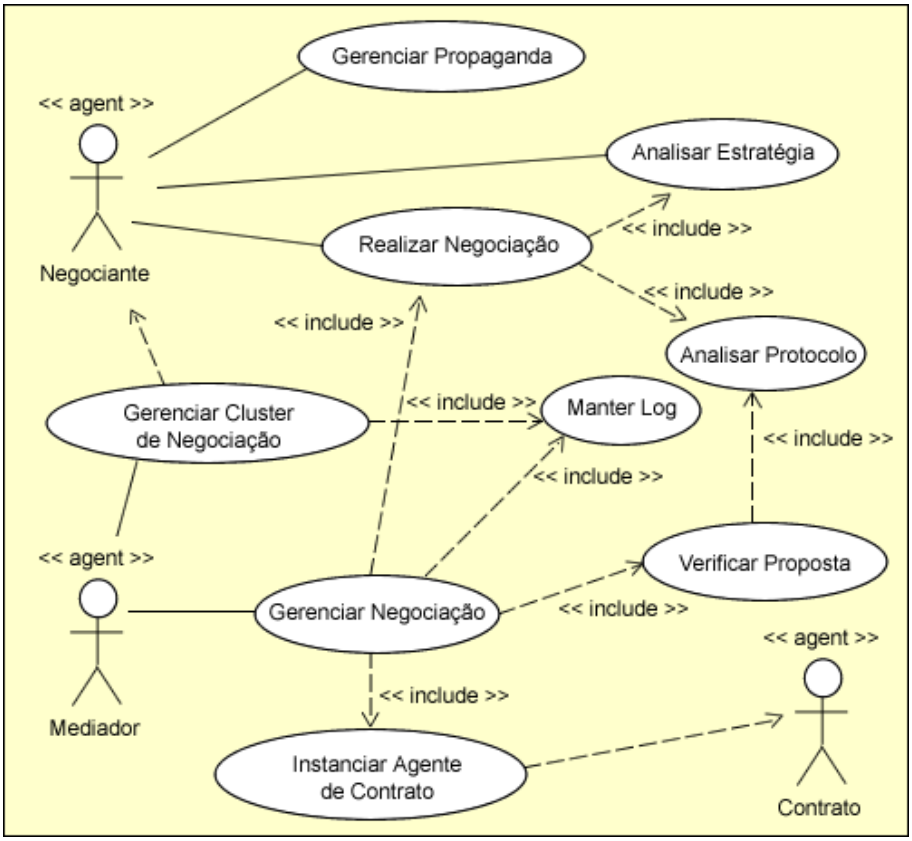

Figura 11: Negociação (diagrama de casos de uso)

Em seguida, detalhamos os principais casos de uso relacionados ao agente mediador. Para cada caso de uso, elaboramos um diagrama de seqüência com a notação AUML, que demonstra a troca de comunicações entre os agentes envolvidos, no sentido de prover uma melhor compreensão das atividades necessárias no processo de negociação. Encontram-se também em anexo os

cenários alternativos (um estudo das possíveis soluções para erros ou desvios de procedimentos do cenário básico) dos casos de uso do agente mediador, bem como os casos de uso dos agentes negociantes.

É importante observar que, em alguns dos diagramas de seqüência que se seguem, optamos por não demonstrar o registro das mensagens recebidas pelo mediador no *log*. Da mesma forma, em caso de desistência de algum negociante, a mensagem do mediador avisando os demais agentes da desistência e da quantidade de agentes ainda em negociação, não é demonstrada.

Caso de Uso: Gerenciar Cluster de Negociação
Ator: agente mediador

Descrição: o agente mediador convoca os agentes negociantes do *cluster* de negociação formado pelo agente *matchmaker*, para que se desloquem até o *marketplace*, onde ocorrerá a negociação, ou os informa sobre o fim da negociação e os manda de volta ao local de origem.

Pré-condição: o agente mediador, o *marketplace* e os agentes negociantes devem ter sido criados, e o mediador deve ter conhecimento da localização dos agentes negociantes que compõem o *cluster*.

Cenário básico:

a) O agente mediador envia mensagens de convocação ou encerramento para os negociantes do *cluster*, para que se dirijam ao *marketplace* ou ao seu local de origem;
b) As mensagens são gravadas num repositório de *log*;
c) O agente negociante se desloca para o local designado (*marketplace* ou origem);
d) O agente negociante atualiza o repositório de propagandas, informando que está participando de uma negociação ou voltou para a origem (ver caso de uso gerenciar propaganda, em anexo);
e) Se o negociante recebeu mensagem de convocação:

i) O agente negociante avisa o mediador de sua chegada, ficando pronto para iniciar a negociação;

ii) O agente mediador verifica se o *cluster* de negociação está completo.

Pós-condição: os agentes do *cluster* de negociação ou estão presentes no *marketplace* e prontos para iniciar o processo de negociação ou voltaram para seus locais de origem e se encontram prontos para nova negociação.

Na Figura 12 apresentamos o diagrama de seqüência referente ao caso de uso gerenciar *cluster* de negociação.

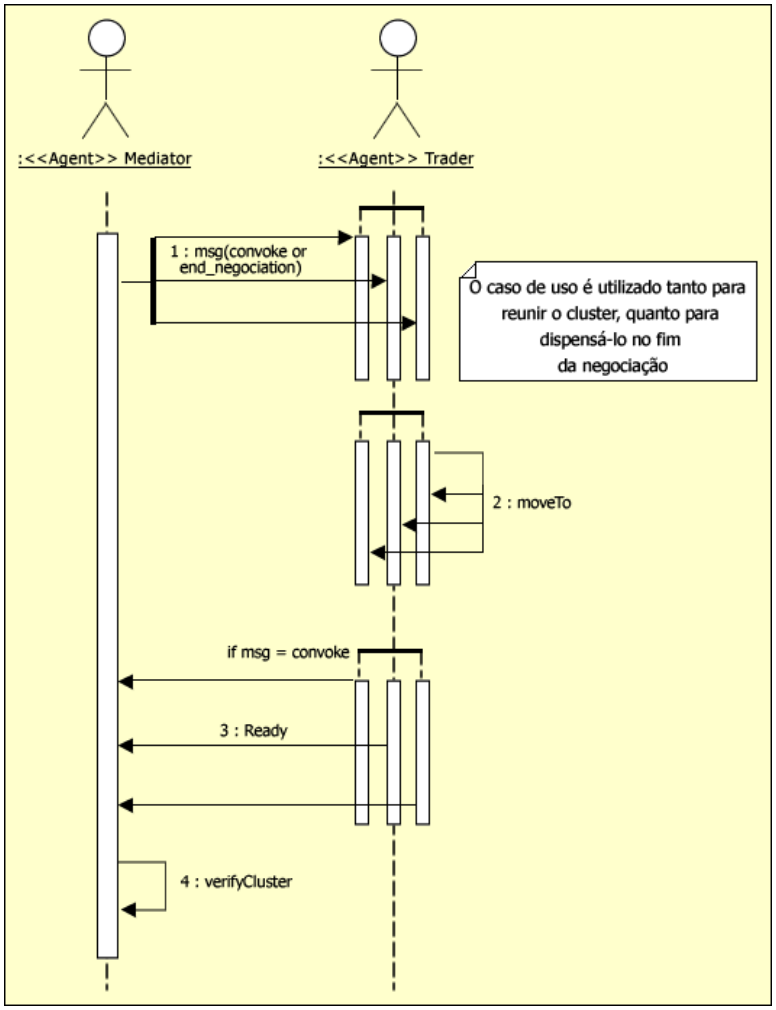

Figura 12: Gerenciar *cluster* de negociação (diagrama de seqüência)

Caso de Uso: Gerenciar Negociação
Ator: agente mediador

Descrição: o agente mediador inicia o processo de negociação pedindo as propostas dos agentes negociantes de acordo com o protocolo de negociação. Os negociantes elaboram suas propostas de acordo com suas respectivas estratégias e as enviam ao agente mediador. Este verifica se as propostas seguem o protocolo de negociação, registra-as no *log* e as repassa aos respectivos agentes negociantes, que verificam se as propostas satisfazem suas necessidades e respondem de acordo.
O agente mediador verifica se um acordo é formado, se inicia outra rodada de negociação, ou ainda, se encerra o processo de negociação quando não há mais acordos possíveis. No caso de um acordo, ele avisa a contraparte sobre o acordo e instancia um agente de contrato para que ele cuide da formalização do negócio. Para iniciar outra rodada de negociação, envia novos pedidos de propostas. Se detectar que nenhum acordo é possível, avisa os agentes negociantes sobre o fim das negociações. Em todos os casos, as respectivas mensagens são registradas no *log*.

Pré-condição: os agentes negociantes devem estar no *marketplace* e prontos para iniciar o processo de negociação.

Cenário básico:

a) O agente mediador solicita aos agentes negociantes propostas de negociação de acordo com o protocolo de negociação;
b) O agente negociante elabora sua proposta de acordo com sua estratégia (ver caso de uso realizar negociação, em anexo);
c) O agente negociante envia a proposta ao agente mediador;
d) O agente mediador verifica se as propostas recebidas estão de acordo com o protocolo de negociação;
e) O agente mediador envia as propostas ao respectivo agente negociante;
f) O agente mediador registra as propostas no repositório de *log* (ver caso de uso manter *log*);

g) O agente negociante verifica se as propostas recebidas satisfazem suas necessidades (ver caso de uso realizar negociação, em anexo);
h) O agente negociante envia ao mediador a sua resposta às propostas recebidas;
i) O agente mediador registra a resposta do negociante no repositório de *log* (ver caso de uso manter *log*);
j) O agente mediador verifica a resposta, para checar se um acordo foi atingido ou se não há mais negociações possíveis. Em caso negativo, ele solicita nova rodada de propostas aos agentes negociantes (Item a);
k) Se um acordo é atingido, o agente mediador:
    i) Avisa o respectivo agente negociante sobre o acordo;
    ii) Registra o aviso no repositório de *log*; e
    iii) Instancia um agente de contrato para formalização do acordo;
l) O agente mediador avisa todos os outros agentes negociantes sobre o fim da negociação.

Pós-condição: negociação realizada com sucesso entre dois agentes negociantes e um agente de contrato criado. Todos os outros agentes negociantes dispensados.

A Figura 13 apresenta o diagrama de seqüência referente ao caso de uso gerenciar negociação.

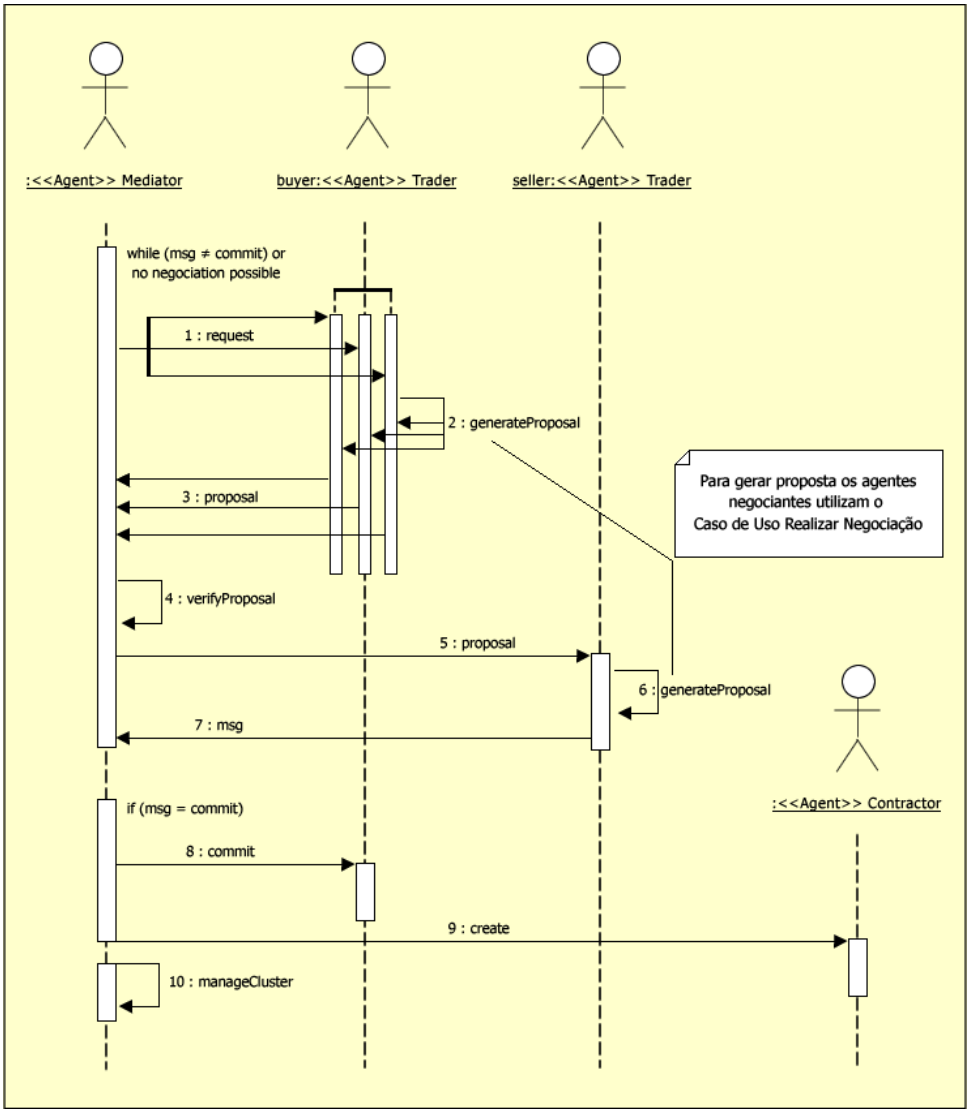

Figura 13: Gerenciar negociação (diagrama de seqüência)

Caso de Uso: Verificar Propostas

Ator: agente mediador

Descrição: o agente mediador recebe proposta de um agente negociante (comprador ou vendedor). Verifica se a mensagem está de acordo com o protocolo de negociação e, em seguida, verifica se a proposta já se encontra no *log* do sistema (se é repetida ou não). Se a mensagem estiver OK, repassa a proposta a outro agente negociante (contraparte). Caso contrário, solicita nova proposta ou dispensa o agente.

Pré-condição: os agentes negociantes devem estar no *marketplace* e o agente mediador deve ter solicitado propostas a eles.

Cenário básico:

a) O agente mediador recebe proposta do agente negociante;
b) De posse da proposta, o agente mediador consulta o protocolo de negociação para verificar se a mensagem está de acordo;
    i) Caso a proposta não se encontre de acordo com o protocolo, é enviado ao agente negociante um novo pedido de proposta;
c) O agente mediador verifica se a proposta já foi feita anteriormente pelo mesmo agente negociante, consultando o *log* (proposta repetida);
    i) Caso a proposta já se encontre no *log*, o mediador envia ao negociante uma mensagem comunicando o fim da negociação para o mesmo;
d) O agente mediador repassa a proposta ao respectivo agente negociante, registrando-a no *log*.

Pós-condição: propostas validadas e registradas no *log*.

Na Figura 14 apresentamos o diagrama de seqüência referente ao caso de uso do agente mediador verificar propostas.

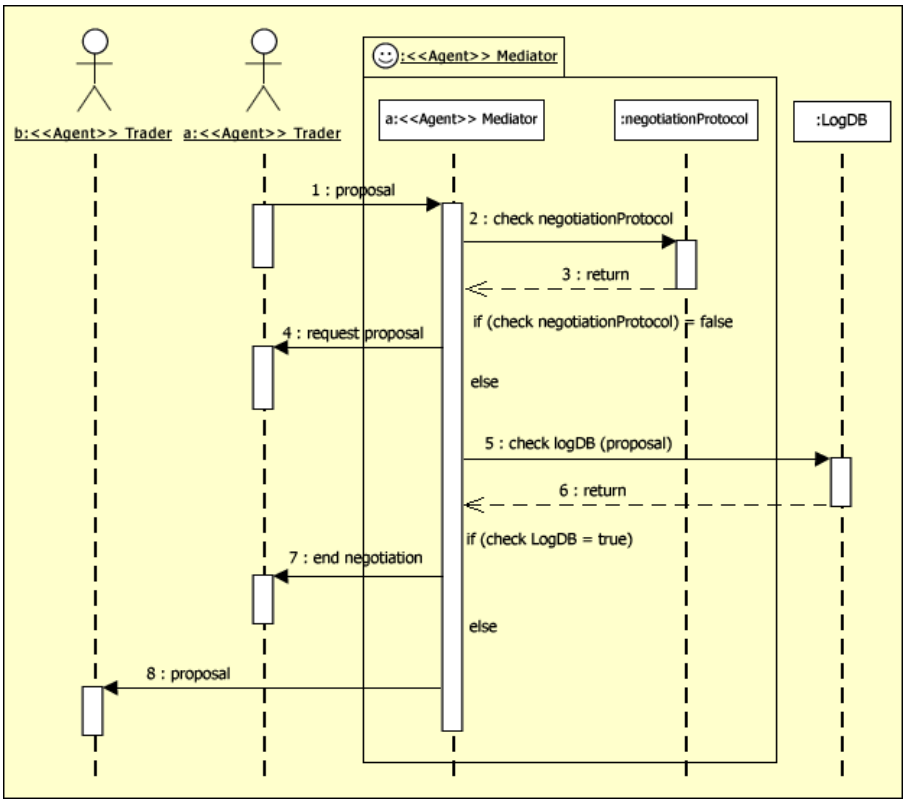

Figura 14: Verificar propostas (diagrama de seqüência)

Caso de Uso: Manter *Log*
Ator: agente mediador

Descrição: o agente mediador deve cadastrar todas as mensagens trocadas com os agentes negociantes durante o desenrolar de uma negociação num repositório de *log*.

Pré-condição: o agente mediador deve ter uma mensagem para registrar.

Cenário básico:

a) O agente mediador envia a mensagem a ser inserida no repositório de *log*;
b) O repositório responde com a confirmação de gravação da mensagem.

Pós-condição: a propaganda é atualizada no repositório.

A Figura 15 apresenta o diagrama de seqüência referente ao caso de uso do agente mediador manter *log*.

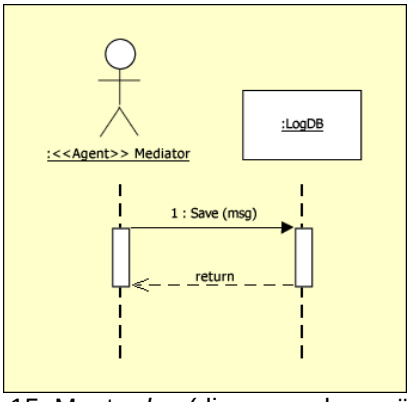

Figura 15: Manter *log* (diagrama de seqüência)

5.4 Diagrama de classes

Dando continuidade à modelagem, apresentamos, na Figura 16, o diagrama de classes, no qual identificamos as principais classes que implementam os agentes relacionados à negociação. Podemos também observar, através do diagrama, algumas classes utilizadas para se obter acesso aos recursos (repositórios) necessários aos agentes para que eles alcancem seus objetivos. A classe que implementa o agente de contrato não apresenta no diagrama seus métodos, por não estar diretamente ligada à fase de negociação.

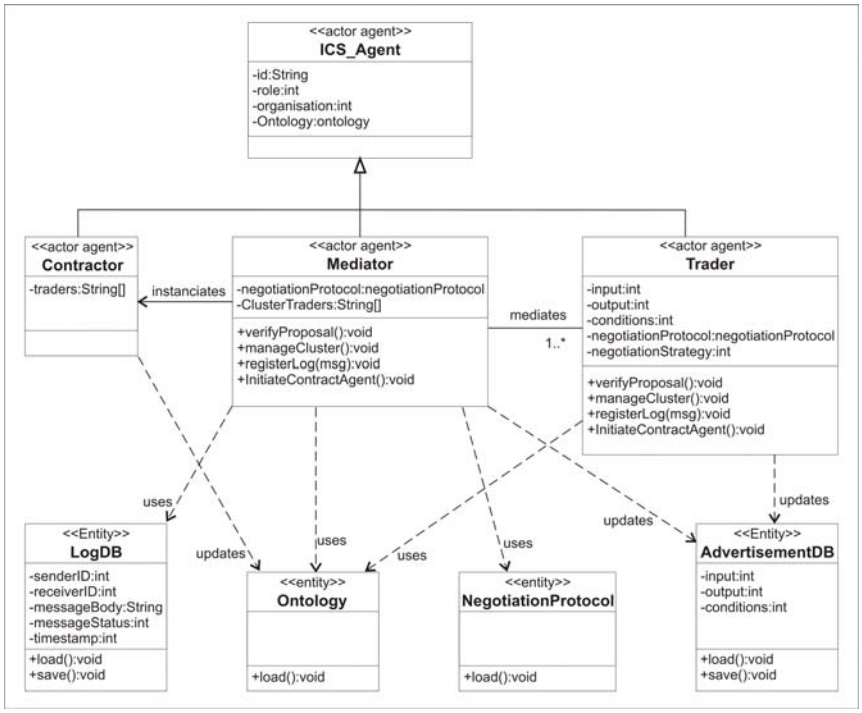

Figura 16: Diagrama de classes

5.5 Conclusão

Neste capítulo, apresentamos a modelagem do agente mediador, que faz parte do processo de negociação como gerenciador das interações ocorrentes entre os agentes negociantes que buscam alcançar seus objetivos dentro do sistema (compra ou venda de bens e serviços).

O agente mediador controla o fluxo de mensagens entre os agentes negociantes, impõe o protocolo de negociação (conjunto de regras que a regem) e, quando um acordo é estabelecido entre os agentes, instancia um agente de contrato que fica responsável pela formalização de um contrato com efeito legal.

Essa modelagem serve como subsídio para construção de um protótipo do agente mediador que será abordado no capítulo 6 deste trabalho. Este protótipo servirá para demonstrar algumas das funcionalidades do agente mediador, além de evidenciar sua viabilidade como gerenciador do processo de negociação dentro do ambiente ICS.

6 PROTOTIPAGEM DO AGENTE MEDIADOR

Neste capítulo, apresentamos a prototipagem do agente mediador. Iniciamos com a proposição de um modelo de ontologia de negociação que servirá para nortear uma negociação experimental do tipo leilão. Em seguida, faremos uma breve descrição das ferramentas utilizadas no desenvolvimento do protótipo e o apresentaremos.

6.1 Introdução

Uma das inovações do ICS é justamente prover conhecimento compartilhado e de fácil manutenção para os agentes do sistema. Isso é obtido através do uso de ontologias, que podem ser armazenadas em repositórios para posterior consulta pelos agentes.

A ontologia da negociação, na qual está contido o protocolo de negociação, é uma dessas ontologias. Nela encontramos os conceitos comuns a todos os tipos de negociação e também as regras que deverão ser impostas pelo mediador para controlar o desenvolvimento da mesma. Ela é utilizada para reger os agentes em negociação, facilitando o uso dos protocolos que são utilizados para atender a diversos tipos de negociação. A principal vantagem deste tipo de solução é a flexibilidade, pois é possível a implementação de vários tipos de negociação sem que seja necessária a codificação dos protocolos, referentes a esses, dentro dos agentes.

Para a implementação do protótipo, desenvolvemos um protocolo capaz de gerenciar uma negociação do tipo leilão. Onde é possível ter diversos compradores realizando propostas para adquirir de um vendedor algum bem ou serviço.

Usamos as plataformas JADE (*Java Agent Development Framework*) e JESS (*Java Expert System Shell*) para desenvolver o nosso protótipo com o intuito de evidenciar a viabilidade do agente mediador em gerenciar uma negociação.

6.2 Ontologia de Negociação

Uma ontologia pode ser compreendida como uma descrição formal de classes de conceitos e seus relacionamentos que descrevem um domínio de aplicação. Esses conceitos possuem propriedades e restrições. De forma geral, desenvolver uma ontologia é definir as classes de conceitos, ordenar essas classes em uma hierarquia (classes e subclasses), definir suas propriedades e preenchê-las com valores permitidos a elas, gerando assim, instâncias desses conceitos (NOY; McGUINNESS, 2001). Recentemente, as ontologias tem sido adotadas em muitas comunidades científicas e de negócios, como modo de compartilhar e reutilizar conhecimento dentro de um dado domínio.

Usualmente agentes representando seus usuários em negociações automatizadas possuem as regras da negociação (protocolo) codificadas internamente (*hard-coded*) nos agentes. Isto constitui claramente uma limitação, pois os agentes idealmente, devem ser capazes de adotar variadas formas de negociação.

A proposta do ICS difere disso, uma ontologia da negociação fica a disposição dos agentes participantes da negociação (negociantes e mediador) em uma *Uniform Resource Identifier* (URI) para que os mesmos possam consultá-la quando for necessário. E dessa forma, é possível disponibilizar vários protocolos para diversos tipos de negociação.

A ontologia da negociação provê um vocabulário básico comum que os negociantes e o mediador devem compartilhar para dar andamento à negociação. Os agentes devem possuir um entendimento comum de como agir durante o processo de negociação, entendendo quando e como enviar propostas ou contrapropostas, e assim, serem capazes de alcançar um acordo.

Os diversos tipos de negociações possuem conceitos comuns a todos eles, mas também possuem especificidades que os diferem. Toda negociação possui compradores, vendedores, um bem a ser negociado, etc. Mas cada tipo de

negociação possui também suas próprias regras que estabelecem o andamento da negociação, que é o protocolo de negociação propriamente dito (TAMA et al., 2002).

Sendo assim, propomos uma modelagem inicial da ontologia da negociação do ambiente ICS, que além de possuir os conceitos comuns a toda negociação, possui também as regras necessárias a cada tipo específico de negociação que os agentes possam vir a participar.

Na Figura 17 demonstramos através de um modelo de entidade-relacionamento os principais conceitos e seus relacionamentos da ontologia da negociação proposta para o ICS. Em anexo encontra-se o código dessa ontologia em *Web Ontology Language* (OWL).

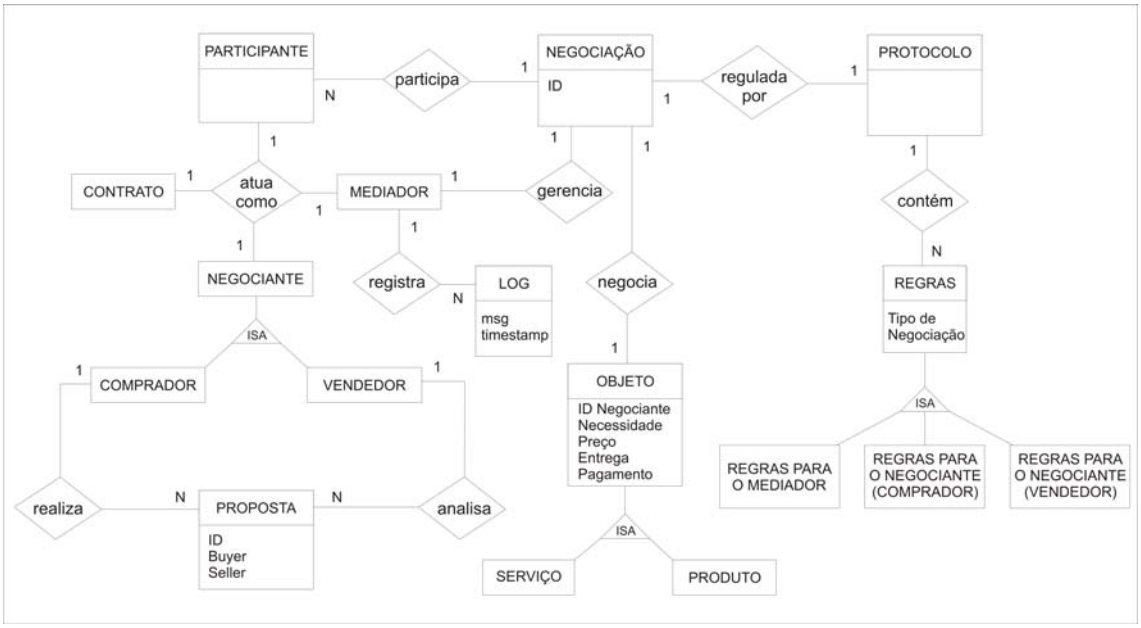

Figura 17: Modelo ER da ontologia da negociação

No modelo proposto, alguns dos conceitos que definem a ontologia da negociação para o ambiente ICS são:

- **Participante:** conceito que descreve o agente que participa da negociação. Pode ser um agente de contrato, negociante ou mediador. Sendo que para cada negociação existe apenas um agente mediador.

- **Objeto:** descreve o bem a ser negociado. Pode ser um serviço ou produto e possui como propriedades: preço, prazo de entrega, forma de pagamento, etc. Um objeto, pode ser composto, por exemplo, de um lote de 1000 unidades de uma determinada publicação.

- **Protocolo de negociação:** conceito que define um protocolo comum a todos os tipos de negociação. É o conjunto de regras que governam uma negociação.

- **Regras:** descreve as ações que devem ser tomadas pelos agentes em função do tipo de negociação estabelecida. São aplicadas de conformidade aos eventos que podem ocorrer na negociação. São compostas, por exemplo, pela regra que estabelece o fim da negociação, pela que estabelece o acordo entre as partes, pela que determina quem deve fazer o quê, etc.

- **Proposta:** conceito que estabelece o que está sendo ofertado ou contra-ofertado pelos agentes que estão em busca de alcançar um acordo.

- ***Log*:** conceito que descreve o espaço para armazenamento de mensagens trocadas entre os agentes durante o processo de negociação.

Como podemos observar, apesar de existirem conceitos comuns a todos os tipos de negociação (participante, proposta, objeto, etc.), através da instanciação das regras, pode-se chegar à definição de um protocolo específico para cada tipo de negociação.

Estas regras devem ser expressas de forma que os agentes possam fazer uso delas. No ambiente ICS elas poderiam, por exemplo, ser codificadas JESS e representadas em OWL dentro da ontologia da negociação. Permitindo assim, a realização de consultas e a inferência das ações que possam vir a ser tomadas pelos agentes dentro de um tipo específico de negociação.

Vale lembrar, que isto é um estudo inicial e que os conceitos propostos para a ontologia da negociação não se limitam àqueles apresentados. Outros estudos

devem ser desenvolvidos para que se estabeleça uma melhor compreensão de como modelar o protocolo de negociação dentro de uma ontologia. Ou mesmo, se é necessário adicionar outros conceitos ao modelo apresentado, para que se aprimore as iterações dos agentes dentro das condições determinadas por um protocolo.

6.3 Protocolo de Negociação

Inicialmente devemos fazer uma diferenciação entre o protocolo de negociação e as estratégias internas de cada agente. O protocolo de negociação determina o fluxo de mensagens entre os participantes e as condições que eles devem seguir durante a negociação. É, essencialmente, de uso compartilhado, isto é, todos os participantes devem estar a par do protocolo. A estratégia interna dos agentes, de modo contrário, é de uso privado de cada agente. Determinando como o agente deve agir, dentro das condições do protocolo, para alcançar um melhor retorno possível para o seu usuário.

Como dito anteriormente, o protocolo de negociação é um conjunto de regras que determina como os agentes participantes de uma negociação interagem entre si, determinando quem pode, quando pode e o quê pode fazer durante este processo.

Como cada tipo de negociação possui suas particularidades, e consequentemente, fluxos de procedimentos diferentes, temos então a necessidade de estabelecer protocolos específicos para cada tipo de negociação (compra direta, leilão, concorrências, etc.).

Para a construção do nosso protótipo, desenvolvemos um protocolo de negociação para o gerenciamento de um leilão que será abordado a seguir.

6.3.1 Leilão Inglês

Leilão é um tipo de negociação no qual existem um ou mais vendedores, diversos compradores e um leiloeiro para gerenciar a negociação. O leilão se desenvolve a partir da realização de ofertas (lances) feitas pelos compradores que desejam adquirir o objeto que está sendo leiloado. As ofertas se sucedem até que se estabeleça um preço que seja aceito pelo vendedor, caracterizando assim, um acordo.

Existem diversas modalidades de leilão, dentre elas destacam-se o leilão holandês (leilão reverso) e o leilão inglês (leilão direto). O leilão holandês é aquele no qual o leiloeiro inicia pedindo um preço de venda elevado e faz ofertas aos compradores com preços cada vez menores, até que alguém aceite a oferta ou o preço de reserva (preço mínimo aceitável pelo vendedor) seja atingido.

No leilão inglês, o leiloeiro inicia informando aos compradores o preço de reserva. Os compradores, então, fazem lances cada vez maiores até que ninguém mais queira realiza lances. O objeto então, é vendido para o comprador que fez o lance de maior valor.

Para o desenvolvimento do protótipo propomos um protocolo de negociação que seja capaz de gerenciar um leilão inglês. Onde possamos ter um vendedor, diversos compradores e o mediador para gerenciar o processo.

Abaixo listamos as regras que gerenciam esse tipo específico de negociação, dentro das condições necessárias ao desenvolvimento do protótipo. Posteriormente elas serão codificadas em JESS (gerando um arquivo chamado AuctionProtocol.clp) para que possam ser utilizadas pelos agentes JADE. As regras estão organizadas de acordo com o agente que está sendo requisitado por uma determinada mensagem, quando essa chega até ele.

Mediador

- Toda mensagem recebida ou enviada deve ser armazenada no log;
- Ao receber uma mensagem do vendedor solicitando a saída da negociação, deve dispensar o cluster e encerrar a negociação;
- Ao receber uma mensagem de um comprador solicitando a saída da negociação, deve dispensar o solicitante e informar os demais participantes da saída do mesmo;
- Quando um comprador for dispensado do cluster, deve verificar se existe número suficiente de participantes para continuar a negociação;
- Não sendo possível continuar a negociação (cluster igual a um comprador e um vendedor), deve verificar se comprador com melhor proposta está presente no cluster.
- Não sendo possível continuar a negociação e estando o comprador com a melhor proposta presente no cluster, deve informar o comprador e vendedor do acordo e instanciar o agente de contrato;
- Não sendo possível continuar a negociação e não estando o comprador com a melhor proposta presente no cluster, deve dispensar o cluster e encerrar a negociação;
- Ao receber mensagem com proposta de um comprador, deve enviar para o vendedor;
- Ao receber do vendedor mensagem rejeitando proposta do comprador, deve repassar a mensagem ao comprador que fez a proposta rejeitada;
- Ao receber do vendedor mensagem informando que determinada proposta é a melhor proposta em um dado momento, deve repassar a mensagem ao comprador que fez a proposta;
- Se o comprador não responder em *x* segundos a uma mensagem, deve ser reenviado uma nova mensagem a ele;

- Se o comprador não responder em *x* segundos a uma mensagem pela segunda vez, deve ser dispensado do cluster e os demais participantes informados da sida do mesmo.

Negociante (comprador)

- Ao receber uma mensagem de dispensa, deve sair do cluster e retorna a sua origem;
- Ao receber uma mensagem informando que outro comprador saiu do cluster, deve atualizar sua estratégia interna;
- Ao receber mensagem rejeitando proposta anteriormente feita, deve verificar sua estratégia interna para determinar se continua na negociação;
- Se desejar continuar na negociação, deve enviar nova proposta ao mediador;
- Não desejando continuar na negociação, deve enviar mensagem ao mediador informando que deseja sair da negociação;
- Ao receber mensagem informando que realizou a melhor proposta em dado momento, deve aguardar por nova mensagem;
- Ao receber mensagem de acordo (*agreement*), deve aguardar mensagem do agente de contrato;
- Ao receber mensagem solicitando proposta, deve enviar proposta ao mediador;
- Toda mensagem de proposta deve ter valor superior ao preço de reserva ou superior a melhor proposta em dado momento.

Negociante (vendedor)

- Ao receber mensagem informando que determinado comprador saiu do cluster, deve atualizar sua estratégia interna;

- Ao receber mensagem de acordo (*agreement*), deve aguardar mensagem do agente de contrato;
- Ao receber mensagem com proposta e estando a variável de melhor proposta com valor igual a zero, deve gravar proposta na variável e enviar ao mediador mensagem informando que a proposta é a melhor no momento;
- Ao receber mensagem com proposta e estando a variável de melhor proposta com valor diferente de zero, deve comparar o valor da variável com o valor da proposta.
- Ao receber mensagem com proposta que seja melhor do que a proposta contida na variável de melhor proposta, deve enviar mensagem ao mediador rejeitando a proposta da variável, gravar a nova proposta na variável e enviar mensagem ao mediador informando que a proposta da variável é a melhor no momento;
- Ao receber mensagem com proposta que seja pior do que a proposta contida na variável de melhor proposta, deve enviar mensagem ao mediador rejeitando a proposta;
- Toda mensagem rejeitando alguma proposta deve conter o valor da melhor proposta no dado momento.

6.4 Ferramentas de Desenvolvimento

Para o desenvolvimento do protótipo utilizamos a ferramenta JADE para desenvolver o agente e em seguida utilizamos o JESS como máquina de inferência (base de raciocínio) para disponibilizar ao agente o protocolo de negociação. Abaixo descreveremos rapidamente cada uma destas ferramentas.

6.4.1 JADE

O JADE é uma plataforma desenvolvida pelo TILAB (*Telecom Italia Labs*) que tem por objetivo simplificar e facilitar o desenvolvimento de aplicações multiagentes conforme as especificações da FIPA. Dentre os serviços oferecidos pela plataforma temos: serviços de nomes (*naming service*), páginas amarelas (*yellow-page service*), transporte de mensagens (*message transport service*), serviços de codificação e decodificação de mensagens e uma biblioteca de protocolos de interação (padrão FIPA) pronta para ser usada.

Toda a comunicação entre os agentes é feita via troca de mensagens ACL (*Agent Communication Language*) FIPA que são baseadas na teoria do ato da fala e permitem a representação de diferentes intenções de comunicação, tais como requisições, propostas, informes, recusas, etc.

O JADE é totalmente desenvolvido em Java e possui as seguintes características principais (BELLIFEMINE et al., 2003):

- **Interoperabilidade:** o JADE é compatível com as especificações FIPA, consequentemente, os agentes JADE podem interagir entre si, pois os mesmos trabalham dentro dos mesmos padrões.
- **Uniformidade e Portabilidade:** o JADE oferece um conjunto de API's (*Application Programming Interface*) que são independentes da plataforma de rede utilizada ou da versão Java empregada para o desenvolvimento das aplicações.
- **Facilidade de uso:** a complexidade da plataforma é escondida por trás de um simples e intuitivo conjunto de API's.
- **Uso do necessário:** os desenvolvedores não necessitam usar todas as características oferecidas pela plataforma. Características não utilizadas não requerem dos desenvolvedores conhecimentos adicionais sobre as mesmas.

A ferramenta utilizada inicialmente no desenvolvimento do ambiente ICS foi o SOMA (*Security and Open Mobile Agents*), mas com o aparecimento do JADE optamos por esta nova plataforma, pois a mesma oferece mais serviços e possibilidades para o desenvolvimento de agentes com uma menor dependência do ambiente de execução pelo uso da linguagem Java.

O JADE pode simplificar a implementação do sistema oferecendo um *middleware* de acordo com especificações FIPA e um conjunto de ferramentas que auxiliam os processos de *debugging* e execução, além de prover uma interface gráfica amigável para o manuseio dos agentes.

6.4.2 JESS

O JESS (*Java Expert System Shell*) é uma ferramenta totalmente escrita em Java desenvolvida pelo SNL (*Sandia National Laboratories*) para a construção de sistemas especialistas. Um sistema especialista é um tipo de software que faz uso de um conjunto de regras que podem ser aplicadas a um conjunto de fatos em um determinado domínio, no intuito de representar o conhecimento de especialistas humanos.

O JESS utiliza um algoritmo especial chamado Rete que faz o casamento das regras que devem ser aplicadas a determinados fatos. O Rete torna o JESS mais rápido do que um simples conjunto de *if...then* em um *loop* em linguagens procedurais. Além disso, o JESS possui capacidades de encadeamento regressivo (*backward-chaining*) e a habilidade de manipular e raciocinar diretamente sobre os objetos Java e linhas de execução (*threads*) criadas em Java.

O JESS pode ser utilizado como uma linguagem de programação, já que pode acessar todas as classes e bibliotecas Java, ou como uma máquina de inferência (JESS *engine*), ou seja, um conjunto de algoritmos capaz de combinar de maneira eficiente as regras que devem ser aplicadas aos fatos do domínio em uso.

No caso do ICS o JESS está sendo utilizado como uma máquina de inferência. Sendo assim, cada mensagem recebida pelos agentes JADE afirma um fato na estrutura JESS que descreve a mensagem recebida. Isto atribui capacidade de raciocínio aos agentes, permitindo que os mesmos possam, através do uso das regras, inferir quais ações devem ser executadas para alcançar os objetivos do sistema.

6.5 Protótipo do Agente Mediador

O protótipo do agente mediador foi desenvolvido com o propósito de mostrar algumas de suas funcionalidades, e deste modo, evidenciar a viabilidade do agente em gerenciar uma negociação.

Utilizamos as plataformas JADE e JESS para sua construção. O JADE oferece algumas ferramentas que facilitaram a implementação do protótipo. Uma delas é o agente *sniffer*, que permite a visualização da troca de mensagens entre os agentes, e a verificação do teor dessas mensagens. O agente *sniffer* foi utilizado no protótipo para implementar a tarefa de manter o *log* da negociação.

Outra ferramenta é o agente *dummy,* que permite configurar e enviar mensagens ACL para qualquer outro agente. No protótipo, utilizamos alguns agentes *dummy* para realizarem o papel dos agentes negociantes, simulando a troca de mensagens durante o leilão.

Dentre as funcionalidades exercidas pelo mediador, no protótipo implementamos as seguintes troca de mensagens entre o mediador e os negociantes:

- Ao receber uma mensagem de PROPOSE do comprador, o mediador consulta o protocolo e a repassa ao vendedor (demonstrado na Figura 18);

- Ao receber uma mensagem de REJECT, o mediador consulta o protocolo e a repassa ao comprador (anotando em seu controle interno que aquele agente teve sua proposta rejeitada);

- Ao receber uma mensagem de INFORM (best propose), o mediador consulta o protocolo e a repassa ao comprador que realizou a melhor proposta (anotando em seu controle interno que aquele agente é o agente com a melhor proposta no momento).

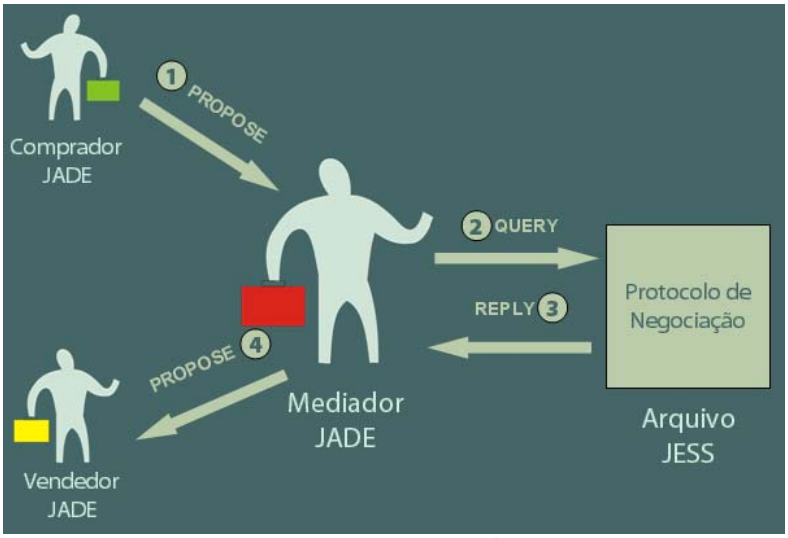

Figura 18: Funcionalidade do protótipo do Mediador

O protótipo ficou então, constituído pelos seguintes componentes:

- **MediatorAgent.java** é um agente Java que inicializa o comportamento JessInterfaceBehaviour que faz o *assert* das mensagens ACL quando elas chegam e usa o JESS como base de raciocínio.

- **AuctionProtocol.clp** é o arquivo JESS que possui as regras que serão utilizadas assim que houver um *match* de fatos incluídos na base de fatos JESS.

- **BasicJessBehaviour.java** é uma classe desenvolvida no *Imperial College of London* que implementa um comportamento JADE que permite carregar uma JESS *engine* dentro do código do agente, possui métodos para converter

mensagens ACL em fatos JESS de tal forma que possam ser incluídos (*assert*) na base de fatos JESS.

6.6 Conclusão

Neste capítulo, propomos uma modelagem inicial da ontologia negociação para o ambiente ICS. Essa ontologia contém os conceitos comuns aos diversos tipos de negociação e o protocolo que deve ser seguido pelos agentes durante um tipo específico de negociação.

Apresentamos um protocolo de negociação (regras) para uma negociação do tipo leilão inglês, que serviu no desenvolvimento do protótipo do agente mediador.

Esse protótipo foi desenvolvido com o intuito de demonstrar algumas das funcionalidades do agente mediador e evidenciar a sua viabilidade em gerenciar uma negociação. Utilizamos o JADE para criar o agente e codificamos o protocolo em JESS para que o agente pudesse posteriormente fazer uso e determinar que ação tomar diante da troca de mensagens durante a negociação.

7 CONCLUSÃO

Nesta dissertação, propusemos o modelo de negociação do ambiente ICS, resultante do refinamento do modelo inicialmente delineado por Fonseca (2003). Utilizamos a metodologia MESSAGE na fase de análise, para obter os diagramas organizacional, de familiaridade, de objetivos e de tarefas, os quais permitiram uma visão mais precisa de como os agentes se organizam e da contextualização do modelo dentro do ambiente, além de contribuírem para um melhor entendimento dos objetivos e tarefas dos agentes que participam do processo de negociação.

Partindo de um melhor entendimento do processo de negociação como um todo, pudemos definir as características desejadas para o agente mediador. Assim, na fase de projeto, utilizando as extensões da AUML, realizamos a modelagem do agente mediador, que é o responsável pela negociação entre os agentes negociantes dentro de um *marketplace* específico.

Dessa forma, desenvolvemos um protótipo do agente mediador, utilizando as plataformas JADE e JESS, como meio de demonstrar algumas das funcionalidades do agente e a sua viabilidade em dar suporte à negociação entre os agentes negociantes. Nossa prototipagem utilizou o leilão inglês como tipo de negociação tratada pelo mediador. Sendo assim, propusemos, também, uma modelagem inicial da ontologia da negociação para o ambiente ICS e um protocolo de negociação específico para esse fim.

Através deste trabalho, observamos a viabilidade do sistema como forma de automatizar a negociação entre agentes inteligentes. O ICS coloca-se, assim, como uma ferramenta de importante potencial dentro da área de comércio eletrônico entre empresas.

Outro aspecto observado com uma clareza maior é que a estrutura do ICS permite uma maior flexibilidade, quando não implementa os protocolos de negociação dentro dos agentes, permitindo que eles trabalhem com diversos tipos

de negociação. Para isso, é necessário apenas que outros protocolos estejam à disposição dos agentes participantes da negociação.

Como proposta para trabalhos futuros, sugerimos:

- A definição de como o agente matchmaker instancia o agente mediador, para que ele possa lidar com processo de negociação;
- Um aperfeiçoamento no processo de negociação para lidar com múltiplos acordos. Permitindo que os agentes negociantes sejam capazes de fechar mais de um acordo por negociação dentro de um mesmo cluster, quando isso for possível em conformidade com o tipo de negociação utilizado;
- A definição de como modelar o protocolo de negociação dentro de uma ontologia, visto que, ontologias são criadas com o propósito de descrever conceitos e seus relacionamentos;
- A definição de novos protocolos de negociação para lidar com outro tipos de negociação: compra direta, leilão reverso, concorrências, etc.;
- A implementação do agente mediador como um todo, juntamente com os agentes negociantes, para que seja finalizada a implementação da fase de negociação dentro do ambiente ICS.

REFERÊNCIAS

ALBERTIN, A. L. **Comércio eletrônico: modelos, aspectos e contribuições de suas aplicações.** São Paulo: Atlas, 2001.

ALMEIDA, Carlos Roberto Baluz. **Composição de *web services* semânticos no ambiente ICS de comércio eletrônico.** Dissertação de Mestrado. Universidade Federal do Maranhão, Curso de Pós-Graduação em Engenharia de Eletricidade, Área de concentração: Ciência da Computação. 2004. São Luis, MA. Brasil.

BARTOLINI, C.; PREIST, C.; JENNINGS, N.R. **A software framework for automated negotiation**. In: F. Giunchiglia, J. Odell, G. Weiß (Ed.) Agent-Oriented Software Engineering III, Springer-Verlag LNCS. 2003.

BASTOS FILHO, Othon de Carvalho. **Modelagem do usuário para o sistema ICS de comércio eletrônico**. Dissertação de Mestrado. Universidade Federal do Maranhão, Curso de Pós-Graduação em Engenharia de Eletricidade, Área de concentração: Ciência da Computação. 2003. São Luis, MA. Brasil.

BELLIFEMINE F.; CAIRE G.; POGGI A.; RIMASSA G. **JADE - A White Paper**. TILAB "EXP in search of innovation" Journal. Volume 3 N. 3, p. 6-19. Itália. 2003.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do usuário**. Rio de Janeiro: Campus, 2000.

CAIRE, G. *et al*. **Agent oriented analysis using MESSAGE/UML**. In: Proc. of the 2nd International Workshop on AgentOriented Software Engineering (AOSE 2001), Montreal, 2001.

CAMERON, Debra. **Eletronic commercer: the new business plataform of the Internet.** Charleston: Computer Tecnology Research Corp., 1997.

CARDOSO, Henrique D. **Sistema multi agente de comércio electrónico**. Tese de Mestrado. Faculdade de Economia, Universidade do Porto. 1999. Porto, Portugal.

CHAVEZ, A.; MAES, P. Kasbah: An Agent MarketPlace for Buying and Selling Goods. In Proceedings of The First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), pp. 75-90, 1996.

EVANS, Richards *et al*. **MESSAGE: Methodology for engineering systems of software agents**. In: EURESCOM – European Institute for Research and Strategic Studies in Telecommunications. Participants in Project P907. 2001.

FONSECA, Luís Carlos Costa. **Sistemas multiagentes para negociação no ambiente ICS de comércio eletrônico**. Dissertação de Mestrado. Universidade Federal do Maranhão, Curso de Pós-Graduação em Engenharia de Eletricidade, Área de concentração: Ciência da Computação. 2003. São Luis, MA. Brasil.

IGLESIAS, C.; GARIJO M.; GONZALES, J.; VELASCO, J.R. **Analysis and design of mutiagent systems using MAS-CommonKADS**. Intelligent Agentes IV: Agent Theories, Architectures and Languages, Lecture Notes in Computer Science LNCS 1365. Springer-Verlag. p. 313-326. 1997.

JENNINGS, Nicholas R. *et al*. **Using intelligent agents to manage business processes**. In: First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96),p.345-360.1996.

LABIDI, Sofiane; FONSECA, Luis Carlos Costa. **Agent mediated e-commerce system**. 2003 IEEE International Conference on Computer System and Applications. Tunis, Tunisia, 2003.

LABIDI, Sofiane *et al*. **Intelligent B2B commerce system**. In: Innovatex/Formatex.(Org.). Techno-legal Aspects of Information Society and New Economy: An Overview. Badajoz, 2003, V. I.

LABIDI, Sofiane; MAIA Jr., Bernardo; MARTINS, Sérgio. **Negotiation Model of the ICS Environment**. In: Third International Workshop on Data Engineering Issues in E-Commerce and Services (DEECS 2007) in conjunction with ACM Conference on Electronic Commerce (EC'07), San Diego, CA, USA, 2007.

LABIDI, Sofiane; MAIA Jr., Bernardo; MARTINS, Sérgio. **Modelo de Negociação do Ambiente ICS**. Workshop – Escola de Sistemas de Agentes para Ambientes Colaborativos (WESAAC 2007), Pelotas, 2007.

NCHO, A.; AIMEUR, E. **Building a multi-agent system for automatic negotiation in web service applications**. In: Conference for Research in Autonomous Agents and MultiAgent Systems (AAMAS'04), New York, USA, 2004.

NOY, Natalya F.; McGUINNESS, Deborah L. **Ontology Development 101: A guide to creating your first ontology**. Knowledge Systems Laboratory Technical Report. 2001.

ODELL, J.; PARANUK, H.v.D.; BAUER, B. **Extending UML for agents**. Proc. of the Agent-Oriented Information Systems, Workshop at the 17th National Conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, AOIS Workshop at AAAI 2000, p.3-17, 2000.

OLIVEIRA, Nathália R. S. **Formação e cumprimento de contratos eletrônicos no sistema de comércio eletrônico - ICS**. Dissertação de Mestrado. Universidade

Federal do Maranhão, Curso de Pós-Graduação em Engenharia de Eletricidade, Área de concentração: Ciência da Computação. 2004. São Luis, MA. Brasil.

OMG – THE OBJECT MANAGEMENT GROUP – Agent Platform Special Interest Group. Agent Technology – Green Paper. Version 1.0. 2000. <http://www.omg.org> Acesso em jan 2007.

TAMA, V.; WOOLDRIDGE, M.; DICKINSON, I. **An ontology based approach to automated negotiation**. AMEC - Agent-mediated Electronic Commerce IV: designing mechanisms and systems, p. 219-237, Itália, 2002.

TOMAZ, Ricardo F. **Uma arquitetura baseada em *web services* semânticos para agrupamento dos agentes negociantes no ambiente ICS de comércio eletrônico**. Dissertação de Mestrado. Universidade Federal do Maranhão, Curso de Pós-Graduação em Engenharia de Eletricidade, Área de concentração: Ciência da Computação. 2003. São Luis, MA. Brasil.

WEISS, Gerhard (ed.). **Multiagent Systems: a modern approach to distributed artificial intelligence**. Cambridge: MIT Press, 1999.

WOOLDRIDGE, Michael; JENNINGS Nicholas R.; SYCARA, Katia. **A roadmap of agent research and development**. Autonomous Agents and Multi-Agent Systems Journal. Volume 1. N.1, p. 7-38. Springer Netherlands: Boston, USA. 1998.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R.; KINNY, D. **The Gaia methodology for agent-oriented analysis and design**. Autonomous Agents and Multi-Agent Systems Journal. N. 3, p. 285-312. Kluwer Academic Press: Netherlands, 2000.

ANEXO 1 – PRINCIPAIS CASOS DE USO DOS AGENTES NEGOCIANTES.

Caso de Uso: Realizar Negociação
Ator: agente negociante

Descrição: O agente negociante recebe uma mensagem do agente mediador, que pode ser um pedido para elaborar uma proposta ou uma proposta para ser avaliada. Em ambos os casos, é gerada uma resposta de acordo com a estratégia que o agente possui para lidar com essas situações. A resposta é então formatada de acordo com o protocolo de negociação e enviada para o agente mediador.

Pré-condição: O agente negociante exista e esteja participando de uma negociação.

Cenário Básico:

a) O agente negociante recebe uma mensagem do agente mediador contendo uma solicitação de proposta ou uma proposta de negociação;
b) O agente negociante utiliza sua estratégia para elaborar sua resposta à mensagem;
c) O agente negociante formata sua resposta de acordo com o protocolo de negociação;
d) O agente negociante envia sua resposta para o agente mediador.

Cenários Alternativos:

a) O agente negociante não recebe a mensagem.
    i) Após algum tempo sem receber mensagem, o agente negociante retorna à sua origem para esperar outra negociação.
b) O protocolo de negociação não está disponível.
    i) O agente negociante tenta novamente até conseguir formatar a resposta adequadamente.
c) O agente mediador não recebe a resposta.
    i) O agente negociante envia novamente a resposta;

ii) Se o erro persistir, tentar avisar o agente mediador para ser excluído do cluster.

Pós-condição: O agente mediador recebe a resposta à sua solicitação.

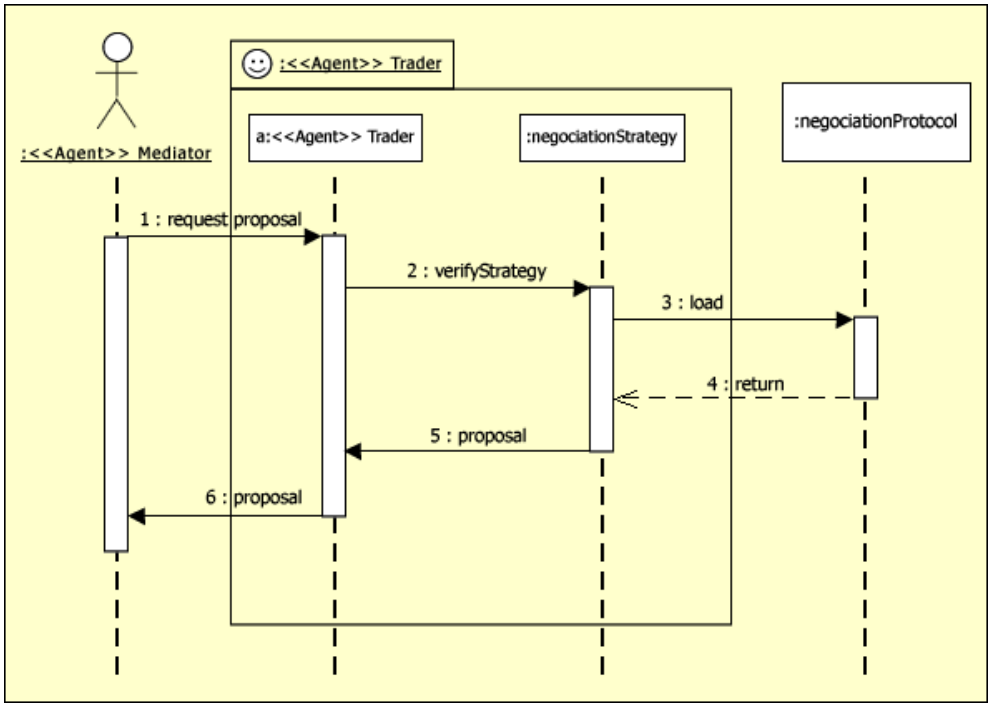

Figura 19: Realizar Negociação (Diagrama de Seqüência).

Caso de Uso: Gerenciar Propaganda

Ator: agente negociante

Descrição: O agente negociante cadastra suas propagandas (necessidades e critérios) no repositório de propagandas, bloqueia-as quando for participar de uma negociação e remove-as ao final de uma negociação bem-sucedida.

Pré-condição: O agente negociante deve ter sido criado pelo usuário.

Cenário Básico:

a) O agente negociante envia a informação a ser inserida no repositório de propagandas;
b) O repositório responde com a confirmação de gravação das informações.

Cenários Alternativos:

a) O agente informa dados em duplicidade ou inconsistentes.
   i) O repositório avisa o agente através de uma mensagem;
   ii) O repositório aguarda uma nova inserção de dados para dar prosseguimento ao caso de uso (Item b do cenário básico).

b) O repositório não confirma a gravação de informações.
   i) O agente envia as informações novamente até obter a confirmação.

Pós-condição: A propaganda é atualizada no repositório.

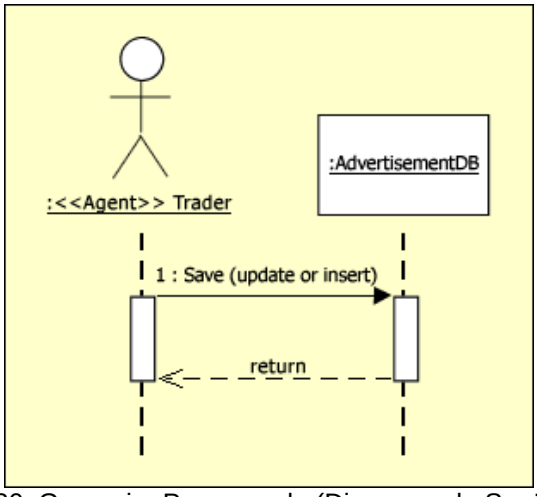

Figura 20: Gerenciar Propaganda (Diagrama de Seqüência).

ANEXO 2 – CÉNARIOS ALTERNATIVOS DOS CASOS DE USO DO AGENTE MEDIADOR

Caso de Uso: Gerenciar Cluster de Negociação

Cenários Alternativos:

a) O agente negociante não recebe a mensagem.
   i) O agente mediador envia a mensagem novamente;
   ii) Se o problema se repetir, descarta o agente do cluster e prossegue com o caso de uso (Item b do cenário básico).

b) As mensagens não são gravadas no repositório de log.
   i) Verificar caso de uso Manter Log.

c) O agente negociante não consegue ou não quer se deslocar.
   i) O agente negociante avisa o agente mediador para ser excluído do cluster.

d) O agente negociante não consegue atualizar o repositório de propagandas.
   i) Verificar o caso de uso Gerenciar Propaganda.

e) O agente negociante não avisa o agente mediador de sua chegada.
   i) O agente negociante envia a mensagem de chegada novamente;
   ii) Se o problema se repetir, o agente mediador descarta o agente do cluster e prossegue com o caso de uso (*timeout*) (Item e.ii do cenário básico).

f) O agente mediador verifica que o cluster de negociação não está completo.
   i) O agente mediador verifica se é possível iniciar a negociação com os agentes negociantes disponíveis;
   ii) Se não for possível, convoca os remanescentes novamente (Item a do cenário básico).

Caso de Uso: Gerenciar Negociação

Cenários Alternativos:

a) Os agentes negociantes não recebem a mensagem de pedido de proposta.
   i) O agente mediador envia a mensagem de proposta novamente;
   ii) Se o problema se repetir, descarta o agente do cluster e prossegue com o caso de uso (Item b do cenário básico).

b) O agente mediador não recebe a mensagem de proposta.

i) O agente negociante envia a mensagem de proposta novamente;
ii) Se o problema se repetir, o agente mediador descarta o agente negociante do cluster e prossegue com o caso de uso (Item c do cenário básico).

c) A proposta do agente negociante não está de acordo com o protocolo de negociação.
i) O agente mediador solicita nova proposta para o agente negociante;
ii) Se o problema se repetir, descarta o agente do cluster e prossegue com o caso de uso (Item c do cenário básico).

d) Os agentes negociantes não recebem as mensagens proposta do agente mediador.
i) O agente mediador envia a mensagem de proposta novamente;
ii) Se o problema se repetir, o agente mediador encerra a negociação (Item l do cenário básico).

e) As mensagens não são gravadas no repositório de log.
i) Verificar caso de uso Manter Log.

f) O agente mediador não recebe a mensagem de resposta.
i) O agente negociante envia a mensagem de resposta novamente;
ii) Se o problema se repetir, o agente mediador encerra a negociação (Item l do cenário básico).

g) A resposta não é gravada no repositório de log.
i) Verificar caso de uso Manter Log.

h) O agente negociante não recebe a mensagem do acordo.
i) O agente mediador envia a mensagem novamente;
ii) Se o problema se repetir, o agente mediador encerra a negociação (Item l do cenário básico).

i) A mensagem do acordo não é gravada no repositório de log.
i) Verificar caso de uso Manter Log.

j) O agente mediador não consegue criar o agente de Contrato.
i) O agente mediador tenta criar o agente de Contrato novamente;
ii) Se o problema se repetir, o agente mediador encerra a negociação (Item l do cenário básico).

k) O agente negociante não recebe a mensagem de fim de negociação.

i) O agente mediador envia a mensagem novamente;

ii) Se o problema se repetir, o agente mediador encerra a negociação (Item l do cenário básico).

Caso de Uso: Verificar Propostas

Cenários Alternativos:

a) O agente mediador não recebe proposta do agente negociante.

i) O agente mediador envia uma nova solicitação de proposta ao agente negociante;

ii) Se o problema se repetir, descarta o agente do cluster e prossegue com o caso de uso.

b) A proposta do agente negociante não se encontra de acordo com o protocolo.

i) O agente mediador solicita uma nova proposta ao agente negociante;

ii) Se o problema se repetir, descarta o agente do cluster e prossegue com o caso de uso.

c) A proposta do agente negociante já se encontra no log.

i) O agente mediador entende que o agente negociante não quer melhorar mais suas propostas e o dispensa do cluster dando continuidade ao caso de uso.

Caso de Uso: Manter Log

Cenários Alternativos:

a) O agente informa dados em duplicidade ou inconsistentes.

i) O repositório avisa o agente através de uma mensagem;

ii) O repositório aguarda uma nova inserção de dados para dar prosseguimento ao caso de uso (Item b do cenário básico).

b) O repositório não confirma a gravação de informações.

i) O agente envia as informações novamente até obter a confirmação.

ANEXO 3 – ONTOLOGIA DA NEGOCIAÇÃO PARA O ICS EM OWL

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="REGRAS_PARA_COMPRADOR">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="REGRAS"/>
    </rdfs:subClassOf>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >REGRAS PARA COMPRADOR</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="NEGOCIANTE">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="PARTICIPANTE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="COMPRADOR">
    <rdfs:subClassOf rdf:resource="#NEGOCIANTE"/>
  </owl:Class>
  <owl:Class rdf:ID="CONTRATO">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PARTICIPANTE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="MEDIADOR">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PARTICIPANTE"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#PARTICIPANTE">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="NEGOCIAÇÃO"/>
    </rdfs:subClassOf>
```

```
</owl:Class>
<owl:Class rdf:ID="PRODUTO">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="OBJETO"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="LOG">
  <rdfs:subClassOf rdf:resource="#NEGOCIAÇÃO"/>
</owl:Class>
<owl:Class rdf:ID="REGRAS_PARA_MEDIADOR">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#REGRAS"/>
  </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >REGRAS PARA MEDIADOR</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="PROPAGANDA">
  <rdfs:subClassOf rdf:resource="#NEGOCIAÇÃO"/>
</owl:Class>
<owl:Class rdf:ID="SERVIÇO">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#OBJETO"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="VENDEDOR">
  <rdfs:subClassOf rdf:resource="#NEGOCIANTE"/>
</owl:Class>
<owl:Class rdf:ID="PROPOSTA">
  <rdfs:subClassOf rdf:resource="#NEGOCIAÇÃO"/>
</owl:Class>
<owl:Class rdf:about="#OBJETO">
  <rdfs:subClassOf rdf:resource="#NEGOCIAÇÃO"/>
</owl:Class>
<owl:Class rdf:ID="REGRAS_PARA_VENDEDOR">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#REGRAS"/>
  </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >REGRAS PARA VENDEDOR</rdfs:label>
</owl:Class>
```

```
  <owl:Class rdf:about="#REGRAS">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="PROTOCOLO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#PROTOCOLO">
    <rdfs:subClassOf rdf:resource="#NEGOCIAÇÃO"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="analyzes">
    <rdfs:domain rdf:resource="#VENDEDOR"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="registers">
    <rdfs:domain rdf:resource="#MEDIADOR"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has">
    <rdfs:domain rdf:resource="#PROTOCOLO"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="participates">
    <rdfs:domain rdf:resource="#PARTICIPANTE"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="makes">
    <rdfs:domain rdf:resource="#COMPRADOR"/>
  </owl:ObjectProperty>
  <owl:FunctionalProperty rdf:ID="id_mediator">
    <rdfs:domain rdf:resource="#MEDIADOR"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID dio Agente Mediador</rdfs:comment>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_buyer">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID do Agente Negociante (comprador)</rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#PROPOSTA"/>
          <owl:Class rdf:about="#COMPRADOR"/>
```

```
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="governs">
    <rdfs:domain rdf:resource="#PROTOCOLO"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="delivery">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Prazo de entrega do objeto negociado</rdfs:comment>
    <rdf:type         rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
    <rdfs:domain rdf:resource="#OBJETO"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="price">
    <rdfs:domain rdf:resource="#OBJETO"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Preço de venda</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_proposal">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID da proposta</rdfs:comment>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#PROPOSTA"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="acts_as">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#CONTRATO"/>
          <owl:Class rdf:about="#MEDIADOR"/>
```

```
          <owl:Class rdf:about="#NEGOCIANTE"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="is_negotiated">
    <rdfs:domain rdf:resource="#OBJETO"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_contractor">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID do Agente de Contrato</rdfs:comment>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#CONTRATO"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="timestamp">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#LOG"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_seller">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#PROPOSTA"/>
          <owl:Class rdf:about="#VENDEDOR"/>
          <owl:Class rdf:about="#OBJETO"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID dio Agente Negociante (vendedor)</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="payment">
```

```
    <rdf:type         rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Forma de pagamento</rdfs:comment>
    <rdfs:domain rdf:resource="#OBJETO"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_negotiation">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID da negociação</rdfs:comment>
    <rdfs:domain rdf:resource="#NEGOCIAÇÃO"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="manages">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#MEDIADOR"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="id_msg">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ID da mensagem</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#LOG"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="announces">
    <rdfs:domain rdf:resource="#NEGOCIANTE"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

ANEXO 4 – CÓDIGOS DO PROTÓTIPO DO AGENTE MEDIADOR

Protocolo de Negociação (AuctionProtocol.clp)

```
;*****************************************************************************
;     English Auction - Negotiation Protocol
;*****************************************************************************
;  Description: negotiation protocol to manage a cluster of traders agents
in english auction.
;*****************************************************************************

; MEDIATOR RULES

(defrule propose-mediator
 ;Quando uma mensagem de 'PROPOSE' chega de um agente ?buyer, esta regra
faz um assert da mensagem recebida para o agente ?seller"
      ?fact <- (ACLMessage (communicative-act PROPOSE) (sender ?buyer)
(receiver ?mediator) (conversation-id ?id) (content ?price))
=>
      (assert (ACLMessage (communicative-act PROPOSE) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (reply-by ?buyer) (content
?price)))
      (retract ?fact)
)

(defrule reject-mediator
      ?fact <- (ACLMessage (communicative-act REJECT-PROPOSAL) (sender
?seller) (receiver ?mediator) (conversation-id ?id) (reply-by ?buyer)
(content ?best_proposal))
=>
      (assert (ACLMessage (communicative-act REJECT-PROPOSAL) (sender
?mediator) (receiver ?buyer) (conversation-id ?id) (content
?best_proposal)))
      (call ?mediator markBuyer (?buyer, "REJECT"))
      (retract ?fact)
)


(defrule inform-best-proposal-mediator
      ?fact <- (ACLMessage (communicative-act ACCEPT-PROPOSAL) (sender
?seller) (receiver ?mediator) (conversation-id ?id) (reply-by ?buyer)
(content ?best_proposal))
=>
      (assert (ACLMessage (communicative-act ACCEPT-PROPOSAL) (sender
?mediator) (receiver ?buyer) (conversation-id ?id) (content
?best_proposal)))
      (call ?mediator markBuyer (?buyer, "BEST"))
      (retract ?fact)
)

(defrule inform-quit-seller
      ?fact <- (ACLMessage (communicative-act CANCEL) (sender ?seller)
(receiver ?mediator) (conversation-id ?id))
=>
      (assert (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (content "DISBAND")))
      (assert (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver (call ?mediator getBuyers() )) (conversation-id ?id) (content
"DISBAND")))
```

```
      (retract ?fact)
)

(defrule inform-quit-buyer
      ?fact <- (ACLMessage (communicative-act CANCEL) (sender ?buyer)
(receiver ?mediator) (conversation-id ?id))
=>
      (assert (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?buyer) (conversation-id ?id) (content "DISBAND")))
      (call ?mediator markBuyer(?buyer, "OUT"))
      (- ?nbuyer 1)
      (assert (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (reply-by ?buyer) (content
"QUIT")))
      (assert (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver (call ?mediator getBuyers() )) (conversation-id ?id) (reply-by
?buyer) (content "QUIT", ?buyer)))
      (if (= ?nbuyer 1)
            (if (= (call ?mediator getBuyers()) (call ?mediator
getBestBuyer()))
                  (bind ?contract (call ?mediator
instantiateContractAgent(?seller, (call ?mediator getBestBuyer()))))
                  (assert (ACLMessage (communicative-act AGREE) (sender
?mediator) (receiver ?seller) (conversation-id ?id) (content ?contract)))
                  (assert (ACLMessage (communicative-act AGREE) (sender
?mediator) (receiver ?buyer) (conversation-id ?id) (content ?contract)))
            else
                  (assert (ACLMessage (communicative-act INFORM) (sender
?mediator) (receiver ?seller) (conversation-id ?id) (content "DISBAND")))
                  (assert (ACLMessage (communicative-act INFORM) (sender
?mediator) (receiver (call ?mediator getBuyers())) (conversation-id ?id)
(content "DISBAND")))
      )
      )
      (retract ?fact)
)


; BUYER RULES

(defrule inform-self-disband
       ?fact <- (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?buyer) (conversation-id ?id) (content "DISBAND"))
=>
 ; mensagem de DISBAND para o ?buyer
 ; o ?buyer deve deixar o cluster
       (call ?buyer moveTo("HOME"))
       (retract ?fact)
)

(defrule inform-buyer-disband
      ?fact <- (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?buyer) (conversation-id ?id) (reply-by ?otherBuyer) (content
"QUIT"))
=>
; mensagem informando DISBAND de outro ?buyer
; o ?buyer deve atualizar sua estratégia
      (call ?buyer updateStrategy(?otherBuyer))
      (retract ?fact)
)
```

```
(defrule reject-buyer
      ?fact <- (ACLMessage (communicative-act REJECT-PROPOSAL) (sender
?mediator) (receiver ?buyer) (conversation-id ?id) (content
?best_proposal))
=>
; o ?buyer deve verificar a sua estratégia
      (if (= (call ?buyer verifyStrategy(?best_proposal)) 1) then
      (assert (ACLMessage (communicative-act PROPOSE) (sender ?buyer)
(receiver ?mediator) (conversation-id ?id) (content (call ?buyer
newProposal(?best_proposal)))))
      else
      (assert (ACLMessage (communicative-act CANCEL) (sender ?buyer)
(receiver ?mediator) (conversation-id ?id))))
      (retract ?fact)
)

(defrule inform-best-proposal-buyer
      ?fact <- (ACLMessage (communicative-act ACCEPT-PROPOSAL) (sender
?mediator) (receiver ?buyer) (conversation-id ?id) (content
?best_proposal))
=>
; o ?buyer deve esparar
      (retract ?fact)
)

(defrule inform-agreement-buyer
      ?fact <- (ACLMessage (communicative-act AGREE) (sender ?mediator)
(receiver ?buyer) (conversation-id ?id) (content ?contract))
=>
; o ?buyer deve esperar por Contract Agent
      (retract ?fact)
)

(defrule propose-buyer
      ?fact <- (ACLMessage (communicative-act CFP) (sender ?mediator)
(receiver ?buyer) (conversation-id ?id) )
=>
      (assert (ACLMessage (communicative-act PROPOSE) (sender ?buyer)
(receiver ?mediator) (conversation-id ?id) (content (call ?buyer
newProposal()))))
      (retract ?fact)
)


; SELLER RULES


(defrule inform-disband-buyer
      ?fact <- (ACLMessage (communicative-act INFORM) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (reply-by ?buyer) (content
"QUIT"))
=>
; mensagem informando DISBAND de um ?buyer
; o ?seller deve atualizar a sua estratégia
      (call ?buyer updateStrategy(?buyer))
      (retract ?fact)
)
(defrule inform-agreement-seller
```

```
      ?fact <- (ACLMessage (communicative-act AGREE) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (content ?contract))
=>
; o ?seller deve esperar por Contract Agent
      (retract ?fact)
)

(defrule propose-seller
      ?fact <- (ACLMessage (communicative-act PROPOSE) (sender ?mediator)
(receiver ?seller) (conversation-id ?id) (reply-by ?buyer) (content
?price))
=>
      (if (= ?best_proposal 0) then
      (bind ?best_proposal ?price)
      (bind ?best_buyer ?buyer)
      (assert (ACLMessage (communicative-act INFORM) (sender ?seller)
(receiver ?mediator) (conversation-id ?id) (reply-by ?buyer) (content
?best_proposal)))
      else
      (if (>= ?best_proposal ?price) then
      (assert (ACLMessage (communicative-act REJECT-PROPOSAL) (sender
?seller) (receiver ?mediator) (conversation-id ?id) (reply-by ?buyer)
(content ?best_proposal)))
      else
      ((assert (ACLMessage (communicative-act REJECT-PROPOSAL) (sender
?seller) (receiver ?mediator) (conversation-id ?id) (reply-by ?best_buyer)
(content ?price)))
      (bind ?best_proposal ?price)
      (bind ?best_buyer ?buyer)
      (assert (ACLMessage (communicative-act ACCEPT-PROPOSAL) (sender
?seller) (receiver ?mediator) (conversation-id ?id) (reply-by ?best_buyer)
(content ?best_proposal))))))
      (retract ?fact)
 )

(defrule send-a-message
      (MyAgent (name ?n))
      ?m <- (ACLMessage (sender ?n))
=>
      (send ?m)
      (retract ?m)
)

(reset)
(run)
```

Agente Mediador (MediatorAgent.java)

```
//Simple Mediator Agent

import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import java.util.*;

public class MediatorAgent extends Agent {

  Hashtable buyers = new Hashtable();

  protected void setup() {
    addBehaviour(new BasicJessBehaviour(this,"AuctionProtocol.clp",1));
```

```
  }

  void markBuyer( ACLAddress agent, String status) {
      buyers.put(agent, status);
  }

  ACLAddress[] getBuyers() {
      return( (ACLAddress) buyers.keys());
  }

  ACLAddress getBestBuyer() {
      Iterator i = buyers.keys();
      while (i.hasNext()) {
            ACLAddress temp = (ACLAddress) i.next();
            if (buyers.get(temp) == "BEST")
                  return (temp);
      }
  }
}
```

Comportamento JADE de *interface* JESS (BasicJessBehaviour.java)

```
/*************************************************************
JADE - Java Agent DEvelopment Framework is a framework to develop multi-
agent systems in compliance with the FIPA specifications.
Copyright (C) 2000 CSELT S.p.A.

GNU Lesser General Public License

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation,
version 2.1 of the License.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the
Free Software Foundation, Inc., 59 Temple Place - Suite 330,
Boston, MA  02111-1307, USA.
*************************************************************/
////////////////////////////////////////////////////////////
//Kaveh Kamyab - Imperial College - 30/10/00
//General Changes
//
//* Replaced all 'ReteException' with 'JessException'
//*  Instantiate  Rete  with  'new  Rete()'  instead  of  'new  Rete(NullDisplay
...)'
//*    Line    193    -    replaced    'FileInputStream    fis    =    new
FileInputStream(jessFile);'
//                with 'FileReader fr = new FileReader(jessFile);'
//* Line 196 - replaced 'Jesp j = new Jesp(fis, jess);'
//                with 'Jesp j = new Jesp(fr, jess);'
//* Replaced all 'stringValue()' with 'stringValue(context)'
//* Replaced '(jess.display()).stderr()'
//                with 'System.err'
```

```
//* Updated getAIDListFromCache to take a ValueVector as parameter. It also
requires Context as a parameter to
//    resolve JESS variables
////////////////////////////////////////////////////////
//Kaveh Kamyab - Imperial College - 30/10/00
//Changes In JessSend
//
//* Replaced 'name()' with 'getName()'
//* Replaced 'engine()' with 'getEngine()'
//* Modified Method 'JessFact2ACL(jess.ValueVector vv)'
//               changed      to      'JessFact2ACL(Context      context,
jess.ValueVector vv)'
//*    Replaced    method    call    accordingly    'JessFact2ACL(vv)'    with
'JessFact2ACL(context, vv)'
//* Replaced 'FunCall.TRUE()' with 'FunCall.TRUE'
//
//* Check for two cases, i.e. if send has a RU.VARIABLE as its first
parameter (send ?m) and secondly if
//send has an RU.FUNCALL as its first parameter (send (assert (ACLMessage
...))).
//1) The first case is straight forward. Get the first parameter of the
ValueVector, extract the Fact Id and find the fact
//by looking up the Fact Id. Pass the resulting ValueVector to
JessFact2ACL(Context context, jess.ValueVector vv).
//2) The second case is a little more tricky. Get the first parameter of
the ValueVector, extract the function call
//(assert). Get the first parameter again and extract the ACLMessage. Jess
variables must be resolved with calls to
//Value.stringValue(context), Value.listValue(context), etc.

import jade.core.*;

import jade.core.behaviours.*;

import jade.lang.acl.ACLMessage;

import jess.*;

import java.io.*;

import java.util.ArrayList;
import java.util.Date;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;


/**
Javadoc documentation for the file
@author Fabio Bellifemine - CSELT S.p.A
@author Kaveh Kamyab - Imperial College of London (upgrade to JESS 5.1)
@version $Date: 2004/09/22 13:08:42 $ $Revision: 2.10 $
*/
/**
 * This is the basic class that implements a behaviour of JADE that allows
 * to embed a Jess engine inside the agent code.
 * <p>
 * <a href="http://herzberg.ca.sandia.gov/jess">Jess</a>
 * supports the development of rule-based expert systems.
 * <p>
```

```
 * This JADE implementation has been tested with version 6.0 of JESS.
 * <p>
 * The programmer can override this class.
 * In particular, its methods <code>ACL2JessString</code> and
 * <code>JessFact2ACL</code> to convert between ACLMessage JADE structure
and
 * Jess facts. Also the method <code>ACLJessTemplate</code> might need to
 * be overide in order to change the deftemplate of the ACLMessage in Jess.
 * <p>
 * When this behaviour is added to the list of agent behaviours,
 * it creates a Jess engine and initializes the engine by:
 * <ul>
 * <li> defining the template of an ACLMessage,
 * <li> defining the userfuntion "send" to send ACLMessages,
 * <li> asserting the fact <code>(MyAgent (name nameofthisagent))</code>,
 * <li> parsing the Jess file passed as a parameter to the constructor.
 * </ul>
 * Then the behaviour loops infinitely by:
 * <ul>
 * <li> waiting that a message arrives,
 * <li> calling the <code>ACL2JessString</code> method that returns the
fact to be
 * asserted in Jess,
 * <li> asserting the fact in Jess,
 * <li> running Jess.
 * </ul>
 * <p>
 * Notice for programmers of the Jess .clp file:
 * <ul>
 * <li> the template of the ACLMessage contains the following slots:
<code>(deftemplate ACLMessage (slot communicative-act) (slot sender)
(multislot receiver) (slot reply-with) (slot in-reply-to) (slot envelope)
(slot conversation-id) (slot protocol) (slot language) (slot ontology)
(slot content) (slot encoding) (multislot reply-to) (slot reply-by))</code>
 * <li> match the fact <code>(MyAgent (name nameofthisagent))</code> to
know the name of your agent;
 * <li> use the userfunction <code>send</code> to send ACLMessages.
 * The parameter of <code>send</code> must be a fact-id of type ACLMessage
or
 * an ACLMessage itself; There are two styles of usage:
 * <p> <code>  ?m <- (ACLMessage (communicative-act cfp) (sender ?s))
 * <br>
 * (send ?m) </code>
 * <p> or, in alternative,
 * <p> <code>(send (assert (ACLMessage (communicative-act cfp)
 * (sender ?s))))</code>
 * <li> remember to load all the Jess Packages you need because, by
default,
 * Jess just loads the built-in functions
 * </ul>
 * <p>
 * Look at the sample file JadeAgent.clp that is shipped with this example.
 * <p>
 * WARNING:
 * FIPA2000 standard has specified an AgentIdentifier to have a
 * template
 * composed of several slots, where the only mandatory one is the agent
name,
 * i.e. the globally unique identifier of the agent (GUID).
 * In order to reduce the porting of the JESS user code, this basic
```

```
 * behaviour automatically
 * takes care of replacing the GUIDs with full-fledged AgentIdentifiers.
 */
public class BasicJessBehaviour extends CyclicBehaviour {
    // class variables
    Rete jess; // holds the pointer to jess
    Agent myAgent; // holds the pointer to this agent
    int m_maxJessPasses = 0; // holds the maximum number of Jess passes for
each run
    int executedPasses = -1; // to count the number of Jess passes in the
previous run
    Hashtable AIDCache; // holds a local cache to map agent names to AID

    /**
     * Creates a <code>BasicJessBehaviour</code> instance
     *
     * @param agent the agent that adds the behaviour
     * @param jessFile the name of the Jess file to be executed
     */
    public BasicJessBehaviour(Agent agent, String jessFile) {
        myAgent = agent;
        AIDCache = new Hashtable();

        // See info about the Display classes in Section 5 of Jess 4.1b6
Readme.htm
        //NullDisplay nd = new NullDisplay();
        // Create a Jess engine
        jess = new Rete();

        // The jess.MiscFunctions is no more used since JESS 6.0 (see e-
mail of Csaba Tenkes
        //try {

//jess.addUserpackage((Userpackage)Class.forName("jess.MiscFunctions").newI
nstance());
        // } catch (Throwable t) { System.out.println(t); }
        try {
            // First I define the ACLMessage template
            jess.executeCommand(ACLJessTemplate());

            // Then I define the myagent template
            jess.executeCommand("(deftemplate MyAgent (slot name))");

            // Then I add the send function
            jess.addUserfunction(new JessSend(myAgent, this));

            // Then I assert the fact (Myagent (name <my-name>))
            jess.executeCommand(
                "(deffacts MyAgent \"All facts about this agent\" (MyAgent
(name " +
                myAgent.getName() + ")))");

            // Open the file test.clp
            FileReader fr = new FileReader(jessFile);

            // Create a parser for the file, telling it where to take input
            // from and which engine to send the results to
            Jesp j = new Jesp(fr, jess);

            // parse and execute one construct, without printing a prompt
```

```
            j.parse(false);
        } catch (JessException re) {
            System.out.println(re);
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }

    /**
     * Creates a <code>BasicJessBehaviour</code> instance that limits
     * the reasoning time of Jess before looking again for arrival of
messages.
     *
     * @param agent the agent that adds the behaviour
     * @param jessFile the name of the Jess file to be executed
     * @param maxJessPasses the maximum number of passes that every run of
Jess
     * can execute before giving again the control to this behaviour;
     * put <code>0</code> if you do not ever want to stop Jess.
     */
    public BasicJessBehaviour(Agent agent, String jessFile, int
maxJessPasses) {
        this(agent, jessFile);
        m_maxJessPasses = maxJessPasses;
    }

    /**
     * executes the behaviour
     */
    public void action() {
        ACLMessage msg; // to keep the ACLMessage

        // wait a message
        if (executedPasses < m_maxJessPasses) {
            System.out.println(myAgent.getName() +
                " is blocked to wait a message...");
            msg = myAgent.blockingReceive();

            // assert the fact message in Jess
            makeassert(ACL2JessString(msg));
        } else {
            System.out.println(myAgent.getName() +
                " is checking if there is a message...");
            msg = myAgent.receive();

            if (msg != null) {
                // assert the fact message in Jess
                makeassert(ACL2JessString(msg));
            }
        }

        // run jess
        try {
            // jess.executeCommand("(facts)");
            if (m_maxJessPasses > 0) {
                executedPasses = jess.run(m_maxJessPasses);
                System.out.println("Jess has executed " + executedPasses +
                    " passes");
            } else {
                jess.run();
```

```
            }
        } catch (JessException re) {
            re.printStackTrace(System.err);
        }
    }

    private boolean isEmpty(String string) {
        return (string == null) || string.equals("");
    }

    /**
      replace a char in a String with a String
      It is used to convert all the quotation marks in backslash quote
      before asserting the content of a message in Jess.
      @return the new String
    */
    private String stringReplace(String str, char oldChar, String s) {
        int len = str.length();
        int i = 0;
        int j = 0;
        int k = 0;
        char[] val = new char[len];
        str.getChars(0, len, val, 0); // put chars into val

        char[] buf = new char[len * s.length()];

        while (i < len) {
            if (val[i] == oldChar) {
                s.getChars(0, s.length(), buf, j);
                j += s.length();
            } else {
                buf[j] = val[i];
                j++;
            }

            i++;
        }

        return new String(buf, 0, j);
    }

    /**
      * makeasserts a fact representing an ACLMessage in Jess. It is called
after the arrival of a message.
      */
    private void makeassert(String fact) {
        try {
            jess.executeCommand(fact);
        } catch (JessException re) {
            re.printStackTrace(System.err);
        }
    }

    /**
     * Remove the first and the last character of the string
     * (if it is a quotation mark) and convert all backslash quote in quote
     * It is used to convert a Jess content into an ACL message content.
     */
    private String unquote(String str) {
        String t1 = str.trim();
```

```
        if (t1.startsWith("\"")) {
            t1 = t1.substring(1);
        }

        if (t1.endsWith("\"")) {
            t1 = t1.substring(0, t1.length() - 1);
        }

        int len = t1.length();
        int i = 0;
        int j = 0;
        int k = 0;
        char[] val = new char[len];
        t1.getChars(0, len, val, 0); // put chars into val

        char[] buf = new char[len];

        boolean maybe = false;

        while (i < len) {
            if (maybe) {
                if (val[i] == '\"') {
                    j--;
                }

                buf[j] = val[i];
                maybe = false;
                i++;
                j++;
            } else {
                if (val[i] == '\\') {
                    maybe = true;
                }

                buf[j] = val[i];
                i++;
                j++;
            }
        }

        return new String(buf, 0, j);
    }

    /**
     * Insert the first and the last character of the string as a quotation
mark
     * Replace all the quote characters into backslash quote.
     * It is used to convert an ACL message content into a Jess content.
     */
    private String quote(java.lang.String str) {
        //replace all chars " in \ "
        return "\"" + stringReplace(str, '"', "\\\"") + "\"";
    }

    /**
     * This method searches in the local cache for the full AID of the
passed agentName.
     * If not found it creates a new AID where only the guid is set.
    **/
```

```
    public AID getAIDFromCache(String agentName) {
        AID result;
        result = (AID) AIDCache.get(agentName);

        if (result == null) {
            result = new AID(agentName);
        }

        return result;
    }

    /**
     * This method searches in the local cache for the full AID of the
passed list of agent names.
     * @param context represents the Rete engine context needed to resolve
the value of JESS variables
     * @param list is a ValueVector of agent names
     * @return a List of AID
     */
    public List getAIDListFromCache(Context context, ValueVector list) {
        ArrayList l = new ArrayList();

        for (int i = 0; i < list.size(); i++) {
            try {
                l.add(getAIDFromCache(list.get(i).stringValue(context)));
            } catch (JessException je) {
            }
        }

        return l;
    }

    /**
     * put a new AID in the local cache.
     * If one exists already with the same agentName, it is overwritten
     */
    public void putAIDInCache(AID aid) {
        AIDCache.put(aid.getName(), aid);
    }

    /** @return a String with the deftemplate command to be executed in
Jess **/
    public String ACLJessTemplate() {
        return  "(deftemplate  ACLMessage  (slot  communicative-act)  (slot
sender)  (multislot  receiver)  (slot  reply-with)  (slot  in-reply-to)  (slot
envelope)  (slot  conversation-id)  (slot  protocol)  (slot  language)  (slot
ontology)  (slot  content)  (slot  encoding)  (multislot  reply-to)  (slot  reply-
by))";
    }

    /**
     *  @return  the  ACLMessage  representing  the  passed  Jess  Fact.  This
message
     * will be then sent by the caller.
     */
    public ACLMessage JessFact2ACL(Context context, jess.ValueVector vv)
        throws jess.JessException {
        // System.err.println("JessFact2ACL "+vv.toString());
        int perf = ACLMessage.getInteger(vv.get(0).stringValue(context));
        ACLMessage msg = new ACLMessage(perf);
```

```
        System.out.println("********      Sender      *********      "      +
vv.get(1).toString());

        if (vv.get(1).stringValue(context) != "nil") {
            msg.setSender(getAIDFromCache(vv.get(1).stringValue(context)));
        }

        if (vv.get(2).toString() != "nil") {
            List          l          =          getAIDListFromCache(context,
vv.get(2).listValue(context));

            for (int i = 0; i < l.size(); i++)
                msg.addReceiver((AID) l.get(i));
        }

        if (vv.get(3).stringValue(context) != "nil") {
            msg.setReplyWith(vv.get(3).stringValue(context));
        }

        if (vv.get(4).stringValue(context) != "nil") {
            msg.setInReplyTo(vv.get(4).stringValue(context));
        }

        //if (vv.get(5).stringValue(context) != "nil")
        //  msg.setEnvelope(vv.get(5).stringValue(context));
        if (vv.get(6).stringValue(context) != "nil") {
            msg.setConversationId(vv.get(6).stringValue(context));
        }

        if (vv.get(7).stringValue(context) != "nil") {
            msg.setProtocol(vv.get(7).stringValue(context));
        }

        if (vv.get(8).stringValue(context) != "nil") {
            msg.setLanguage(vv.get(8).stringValue(context));
        }

        if (vv.get(9).stringValue(context) != "nil") {
            msg.setOntology(vv.get(9).stringValue(context));
        }

        if (vv.get(10).stringValue(context) != "nil") {
            //FIXME undo replace chars of JessBehaviour.java. Needs to be
done better
            msg.setContent(unquote(vv.get(10).stringValue(context)));
        }

        if (vv.get(11).stringValue(context) != "nil") {
            msg.setEncoding(vv.get(11).stringValue(context));
        }

        //System.err.println("JessFact2ACL type is "+vv.get(15).type());
        if (vv.get(12).toString() != "nil") {
            List          l          =          getAIDListFromCache(context,
vv.get(12).listValue(context));

            for (int i = 0; i < l.size(); i++)
                msg.addReplyTo((AID) l.get(i));
        }
```

```
        if (vv.get(13).stringValue(context) != "nil") {
            try {
                msg.setReplyByDate(new Date(Long.parseLong(vv.get(13)

.stringValue(context))));
            } catch (Exception e) { /* do not care */
            }
        }

        return msg;
    }

    /**
     * @return the String representing the facts (even more than one fact
is
     * allowed, but this method just returns one fact)
     * to be asserted in Jess as a consequence of the receipt of
     * the passed ACL Message.
     * The messate content is quoted before asserting the Jess Fact.
     * It is unquoted by the JessFact2ACL function.
     */
    public String ACL2JessString(ACLMessage msg) {
        String fact;

        if (msg == null) {
            return "";
        }

        // I create a string that asserts the template fact
        fact = "(assert (ACLMessage (communicative-act " +
            ACLMessage.getPerformative(msg.getPerformative());

        if (msg.getSender() != null) {
            fact = fact + ") (sender " + msg.getSender().getName();
            putAIDInCache(msg.getSender());
        }

        Iterator i = msg.getAllReceiver();

        if (i.hasNext()) {
            fact = fact + ") (receiver ";

            while (i.hasNext()) {
                AID aid = (AID) i.next();
                putAIDInCache(aid);
                fact = fact + aid.getName();
            }
        }

        if (!isEmpty(msg.getReplyWith())) {
            fact = fact + ") (reply-with " + msg.getReplyWith();
        }

        if (!isEmpty(msg.getInReplyTo())) {
            fact = fact + ") (in-reply-to " + msg.getInReplyTo();
        }

        //if (!isEmpty(msg.getEnvelope()))      fact=fact+") (envelope " +
msg.getEnvelope();
        if (!isEmpty(msg.getConversationId())) {
```

```
            fact = fact + ") (conversation-id " + msg.getConversationId();
        }

        if (!isEmpty(msg.getProtocol())) {
            fact = fact + ") (protocol " + msg.getProtocol();
        }

        if (!isEmpty(msg.getLanguage())) {
            fact = fact + ") (language " + msg.getLanguage();
        }

        if (!isEmpty(msg.getOntology())) {
            fact = fact + ") (ontology " + msg.getOntology();
        }

        if (msg.getContent() != null) {
            fact = fact + ") (content " + quote(msg.getContent());
        }

        if (!isEmpty(msg.getEncoding())) {
            fact = fact + ") (encoding " + msg.getEncoding();
        }

        i = msg.getAllReplyTo();

        if (i.hasNext()) {
            fact = fact + ") (reply-to ";

            while (i.hasNext()) {
                AID aid = (AID) i.next();
                putAIDInCache(aid);
                fact = fact + aid.getName();
            }
        }

        if (msg.getReplyByDate() != null) {
            fact = fact + ") (reply-by " + msg.getReplyByDate().getTime();
        }

        fact = fact + ")))";

        return fact;
    }

    /**
     * This class implements the Jess userfunction to send ACLMessages
     * directly from Jess.
     * It can be used by Jess by using the name <code>send</code>.
     */
    public class JessSend implements Userfunction {
        // data
        Agent my_agent;
        BasicJessBehaviour bjb;

        public JessSend(Agent a, BasicJessBehaviour b) {
            my_agent = a;
            bjb = b;
        }
```

```
        // The name method returns the name by which the function appears
in Jess
        public String getName() {
            return ("send");
        }

        //Called when (send ...) is encountered
        public Value call(ValueVector vv, Context context)
            throws JessException {
            //for (int i=0; i<vv.size(); i++) {
            //     System.out.println("   parameter   "   +   i   +   "="   +
vv.get(i).toString() +
            //   " type=" + vv.get(i).type());
            //  }
            ////////////////////////////////
            // Case where JESS calls (send ?m)
            if (vv.get(1).type() == RU.VARIABLE) {
                //      Uncomment      for      JESS      5.0      vv      =
context.getEngine().findFactByID(vv.get(1).factIDValue(context));
                vv = context.getEngine().findFactByID(vv.get(1)
                                                      .factValue(context)
                                                      .getFactId());
//JESS6.0
            }
            ////////////////////////////////
            // Case where JESS calls (send (assert (ACLMessage ...)))
            else if (vv.get(1).type() == RU.FUNCALL) {
                Funcall fc = vv.get(1).funcallValue(context);
                vv = fc.get(1).factValue(context);
            }

            ACLMessage msg = bjb.JessFact2ACL(context, vv);
            my_agent.send(msg);

            return Funcall.TRUE;
        }
    } // end JessSend class
} // end JessBehaviour
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo