UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO
ACESSO E COMPARTILHAMENTO DE ONTOLOGIAS POR MEIO DE AGENTES DE SOFTWARE

LUCIANO PALMA DA SILVA PEREIRA

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO
ACESSO E COMPARTILHAMENTO DE ONTOLOGIAS POR MEIO DE AGENTES DE SOFTWARE

LUCIANO PALMA DA SILVA PEREIRA

Relatório apresentado ao Instituto de Computação da Universidade Federal de Mato Grosso, para obtenção do título de Bacharel em Ciência da Computação.

CUIABÁ – MT
2014

UNIVERSIDADE FEDERAL DE MATO GROSSO

INSTITUTO DE COMPUTAÇÃO

COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCIANO PALMA DA SILVA PEREIRA

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Ciência da Computação como uma das exigências para obtenção do título de Bacharel em Ciência da Computação da Universidade Federal de Mato Grosso

Aprovado por:

Prof. Dr. Nelcileno Virgílio de Souza Araújo
Instituto de Computação
(ORIENTADOR)

Prof. Dr. Andreia Gentil Bonfante
Instituto de Computação
(PROFESSORA CONVIDADA)

Prof. Msc. Fernando Castilho
(SUPERVISOR)

DEDICATÓRIA

À minha família pelo apoio e incentivo.

Aos meus amigos de infância pelos bons momentos que compartilhamos.

Ao Grupo Escoteiro Uniselva, especialmente aos chefes Célia e Roosevelt, por tudo que tenho aprendido e por me ensinarem a importância de passar adiante o conhecimento que adquirimos ao longo da vida.

AGRADECIMENTOS

À minha família pelo carinho que sempre tivemos uns pelos outros.

Ao meu orientador, prof. Dr. Nelcileno Virgílio, e ao meu Supervisor, prof. Msc. Fernando Castilho, por todo apoio e confiança que me deram.

Aos membros do projeto LAVI-França, por proporcionarem essa oportunidade de aprendizado e contribuição.

À minha namorada, Gisele, pelo apoio e por estar sempre ao meu lado, me incentivando a sempre seguir em frente.

Aos meus colegas da faculdade, por transformarem uma simples “hora do café” em um longo momento de risadas e descontração.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ACC	*Agent Communication Channel*
ACL	*Agent Comunication Language*
AMS	*Agent Management System*
AOP	*Agent Oriented Programming*
API	*Application Programming Interface*
ATA	*AT Attachment*
DDR3	*Double Data Rate nível 3*
DF	*Directory Facilitator*
DTD	*Document Type Definition*
EPL	*Eclipse Public License*
FIPA	*Foundation for Intelligent Physical Agents*
IDE	*Integrated Development Environment*
JENA	*Open source Semantic Web framework for Java*
JFact	*Java port of the FaCT++ OWL DL reasoner*
JVM	*Java Virtual Machine*
LAVI	*Laboratório de Ambientes Virtuais Interativos*
LED	*Light-emitting-diode*
MTP	*Message Transport Protocol*
OOP	*Object Oriented Programming*
OS	*Operating System*
OWL	*Web Ontology Language*
PHP	*PHP: Hypertext Preprocessor*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*

UFMT	*Universidade Federal de Mato Grosso*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*
XPath	*XML Path Language*
XQuery	*XML Query Language*

RESUMO

A natureza deste estágio está no desenvolvimento de uma aplicação multi-agente com acesso a ontologias, unindo os conceitos de Agentes de Software, Web Semântica e Ontologia, servindo de apoio e fonte de estudos para o projeto de pesquisa *LAVI – França*. A aplicação desenvolvida durante o estágio simula o ambiente de uma pizzaria, onde o garçom e os clientes são agentes de software que se comunicam através de trocas de mensagens utilizando ontologias. Este relatório expõe as atividades realizadas na disciplina de Estágio Supervisionado na Universidade Federal de Mato Grosso, no período de Maio a Agosto de 2014, do discente Luciano Palma da Silva Pereira, graduando do curso de Ciência da Computação, realizado no Instituto de Computação da UFMT, com supervisão do Prof. Msc. Fernando Castilho e orientação do Prof. Dr. Nelcileno Virgílio.

INTRODUÇÃO

As atividades realizadas durante o estágio consistem na criação de uma aplicação multi-agente com comunicação entre um agente Garçom e vários agentes Clientes, para acessar e compartilhar recursos contidos em uma ontologia.

O agente Garçom faz acesso a ontologia intitulada *pizza.owl* que possui informações sobre pizzas (base, recheio, nome, tipo, etc.) e repassa essas informações aos agentes clientes de acordo com suas requisições. Para o desenvolvimento dessa aplicação, foi utilizada a IDE Eclipse para a programação na linguagem Java. A criação dos agentes foi feita através do framework para sistemas multi-agentes JADE. Para a criação da ontologia, foi utilizada a ferramenta Protégé Desktop 4.0. O acesso do agente Garçom à ontologia foi realizado com auxílio da API Java OWLAPI e do *reasoner* JFact 1.2.2, capaz de realizar as inferências necessárias para a execução da aplicação. Todos os agentes clientes estabeleceram comunicação com o Garçom e tiveram suas requisições atendidas. Os métodos utilizados demonstraram-se viáveis para o desenvolvimento de um sistema multi-agente sensível ao contexto. As técnicas, métodos e resultados das atividades realizadas estão descritas detalhadamente nesse relatório.

O projeto LAVI-França, do instituto de Computação da UFMT, reúne diversas áreas de conhecimento, também conhecidas como pilares. Esses pilares incluem pesquisas em áreas como Agentes Inteligentes, Web Semântica e Ontologias. O objetivo geral do estágio realizado é adquirir conhecimento e experiência nas áreas citadas, através do desenvolvimento de uma aplicação que possa servir de fonte de pesquisa e estudo para o projeto LAVI-França.

O presente relatório foi organizado da seguinte maneira: na primeira parte é apresentada uma revisão teórica, abordando temas relacionados ao estágio e às disciplinas feitas no curso de Ciência da Computação; na segunda parte, as técnicas, ferramentas e métodos utilizados no desenvolvimento da aplicação são apresentados com detalhes; em seguida, os resultados obtidos são apresentados, demonstrando a viabilidade das técnicas e métodos utilizados; na quarta parte são discutidas as dificuldades encontradas durante a realização do estágio e como elas foram

superadas; por fim, as conclusões sobre o desenvolvimento das atividades do estágio são apresentadas na quinta parte.

1. REVISÃO DE LITERATURA

Para o desenvolvimento da aplicação e deste relatório, foram necessários diversos estudos e pesquisas sobre temas relacionados à ciência da computação. Muitos desses temas ainda não são abordados nas disciplinas do curso de Ciência da Computação, fazendo com que o estágio seja de grande valia para a formação acadêmica do aluno. Nesta seção será realizada uma abordagem teórica desses principais temas e conceitos. Os termos abordados são: Abordagem Orientada a Agente; Tipologia de Agentes; Arquitetura FIPA; Web Semântica; XML e RDF.

1.1 Abordagem Orientada a Agente

Um número crescente de computadores e dispositivos eletrônicos tem sido visto hoje em termos de agentes autônomos, podendo inclusive estabelecer comunicação, consolidando um sistema multi-agentes. A abordagem baseada em agentes é empregada quando uma situação requer que o processamento seja descentralizado e auto-organizado. Para essas situações, os agentes, juntamente com outras tecnologias (banco de dados relacionais, componentes, orientação a objetos, etc.) fornecerão as ferramentas apropriadas para que uma organização obtenha maior benefício dos sistemas baseados em agentes.

Para Wooldridge e Jennings (1995), um agente autônomo pode ser definido como um hardware ou (mais usualmente) um software possui as seguintes propriedades:

- **Autonomia**: agentes operam sem intervenção direta de humanos ou outros e possuem algum tipo de controle sobre suas ações e estado interno;
- **Habilidade Social**: agentes interagem com outros agentes (e possivelmente humanos) através de algum tipo de linguagem para comunicação de agentes;
- **Reatividade**: agentes percebem seus ambientes e respondem em tempo hábil a mudanças que ocorrem nele;

- **Pró-atividade**: agentes não agem simplesmente em resposta ao ambiente, eles são capazes de exibir comportamentos direcionados a objetivos ao tomarem a iniciativa.

Existem diversas maneiras de interpretar o termo “agente” em computação e vários autores trazem definições levemente distintas. Um agente pode ser entendido como um sistema de computador encapsulado, situado em um determinado contexto/ambiente, capaz de realizar ações e alcançar objetivos naquele ambiente de forma autônoma e flexível. Em outras palavras, um agente é um programa de computador que executa, de forma autônoma, ações pré-definidas para alcançar objetivos projetados pelo programador.

Para Odell (2010), um agente é uma entidade autônoma que pode se adaptar e interagir com o ambiente em que se encontra. Agentes podem ser projetados para se comportar de forma hierárquica. Contudo, a força dos agentes encontra-se no fato de poderem ser usados tanto de forma hierárquica quanto colaborativa, ao passo que outras abordagens carecem de tal flexibilidade e escalabilidade.

Odell (2010) aponta ainda que agentes implantados em sistemas de TI devem contar com três propriedades importantes:

- **Autonomia** – é capaz de atuar sem intervenção externa direta;
- **Interatividade –** comunica e colabora com o ambiente e outros agentes;
- **Adaptabilidade –** pode ser projetado para tomar decisões difíceis e até mesmo modificar seu comportamento baseado em sua experiência, ou seja, o agente pode aprender e evoluir.

O conceito de programação orientada a agentes foi primeiramente utilizado por Yoav Shoham, em 1990. De acordo com Shoham, a Programação Orientada a Agentes (AOP) pode ser vista como uma especialização da Programação Orientada a Objetos (OOP). A tabela 1 mostra uma comparação entre eles

.

Tabela 1 – OOP vs. AOP (Shoham, 1990)

	OOP	AOP
Unidade Básica	objeto	Agente
Parâmetros que definem o estado da unidade básica	sem restrições	convicção, compromisso, capacidades, escolhas…
Processo de computação	passagem de mensagem e métodos de resposta	passagem de mensagem e métodos de resposta
Tipo de mensagem	sem restrições	informar, solicitar, propor, garantir, recusar…
Restrições nos métodos	sem restrições	honestidade, consistência…

Cada vez mais companhias adotam agentes juntamente com softwares e sensores para detectar mudanças no ambiente, tomar decisões e executar ações de forma autônoma. O sucesso e benefícios alcançados por essas companhias demonstram as vantagens e contribuem para o crescimento da implantação de sistemas baseados em agentes.

1.1.1 Tipologia de Agentes

Um agente existente pode ser classificado em várias dimensões. Inicialmente, pode-se classificar um agente quanto à sua mobilidade, ou seja, sua capacidade de se mover de um ambiente para outro. Dessa forma, é possível definir as classes de agentes estáticos ou móveis (Nwana, 1998).

Em seguida, pode-se classificá-los com relação ao tipo de comportamento que apresentam: deliberativo ou reativo. Agentes deliberativos possuem um modelo interno de raciocínio simbólico, podendo engajar em planejamentos e negociações com outros agentes a fim de alcançar seus objetivos. Já os agentes reativos não possuem um modelo simbólico do ambiente em que se encontram e, portanto, possuem comportamento do tipo estímulo/resposta, respondendo ao atual estado do ambiente em que se encontram.

Em terceiro lugar, é possível classificar um agente de acordo com os diversos atributos que eles devem exibir. Três desses atributos são: autonomia, aprendizagem, cooperação. Agentes autônomos são pró-ativos e são classificados sob

o princípio de que agentes podem operar por conta própria, sem a necessidade de auxílio humano. A cooperação entre agentes é um atributo chave: é por causa dele que existem sistemas multi-agente. Por último, para que agentes sejam realmente inteligentes, eles precisam aprender enquanto reagem/interagem com o ambiente em que se encontram, de modo que seu desempenho aumenta com o tempo. A Figura 1 apresenta a classificação para essa tipologia de agentes.

Figura 1 – Classificação de Agentes (Nwana, 1998)

À medida que a computação orientada a agentes evolui, novas formas de classificar os agentes surgem para suprir as necessidades e facilitar a implementação de sistemas multi-agente.

1.2 Sistemas Multi-Agentes (MAS)

Sistemas Multi-Agentes representam um poderoso modelo para solucionar problemas computacionais distribuídos. Através deles, é possível utilizar múltiplos agentes de forma colaborativa para resolver problemas quando um único agente não é capaz. Segundo Polad (2007), Sistemas Multi-Agentes, ou MAS, são sistemas distribuídos, compostos por um número de entidades de software autônomas chamadas agentes.

Quando há apenas um agente no cenário, normalmente tem-se a forma mais simples de representação do conhecimento e não se requer técnicas além do comportamento do próprio agente. Porém, quando existem múltiplos agentes no ambiente, é necessário que um agente tenha *conhecimento* dos outros agentes nesse ambiente. Nesse caso, existem duas possibilidades para o comportamento desses agentes: a primeira, onde todos os agentes dividem um *propósito comum* nesse ambiente. Um exemplo é uma sala de aula inteligente onde cada agente opera um

equipamento diferente (projetor, computador, iluminação, etc.). A segunda possibilidade é quando os agentes *não* dividem o mesmo propósito. Como exemplo tem-se um sistema de compra e vendas, no qual o agente comprador tenta comprar um produto pelo menor preço, ao passo que o agente vendedor tenta vender esse produto pelo maior valor possível.

Em sistemas multi-agentes cooperativos, os agentes buscam, através de interações, resolver em conjunto tarefas ou maximizar utilidades. Devido à interação desses agentes, a complexidade desses sistemas pode aumentar rapidamente conforme o número de agentes ou da sofisticação de seus comportamentos (Panait, Luke, 2004).

Apesar da cooperação entre agentes ocorrer na maior parte dos sistemas multi-agentes, é comum os projetistas de sistemas não se importarem se esses agentes estão de fato cooperando ou competindo, desde que eles estejam executando comportamentos de forma coordenada. Agentes podem cooperar através da Linguagem de Comunicação de Agentes (ACL) para auxiliar o compartilhamento de forma rica e bem entendida a respeito da semântica do conteúdo das mensagens e da semântica do contexto da comunicação do conteúdo dessas mensagens.

As Figuras 2 a 7 mostram um exemplo de implantação de um sistema multi-agente na área de *e-commerce*. Apesar de sistemas onde parceiros comerciais negociam diretamente através da Internet levarem a um mercado flexível e mais competitivo, com a implantação de um sistema multi-agente ganha-se autonomia e maior flexibilidade no processo de negociação entre comprador e vendedor.

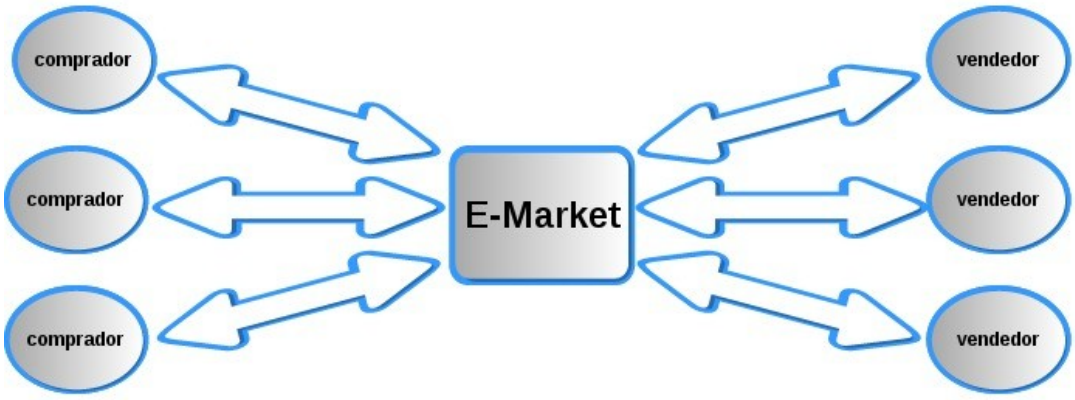

Figura **2** – Sistema e-commerce com negociação direta entre parceiros comerciais

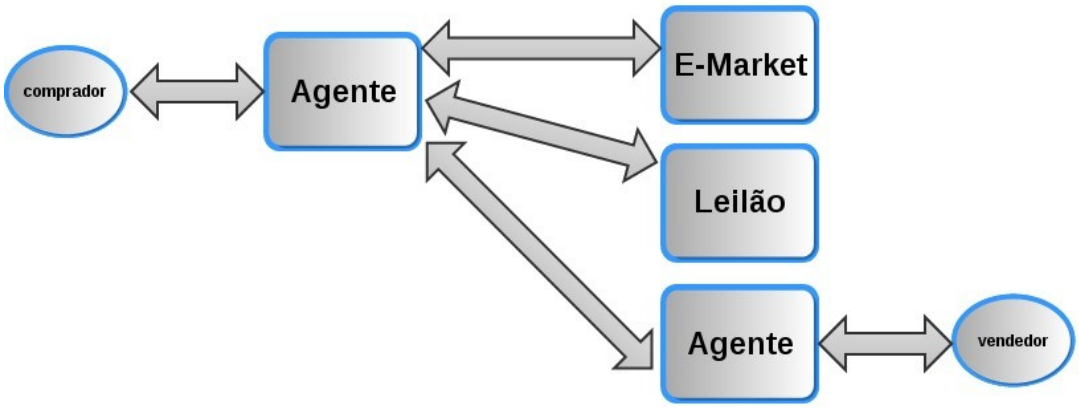

Figura **3** – Sistema e-commerce multi-agente. Agentes cuidam do processo de negociação

A negociação entre agentes envolve, através do estabelecimento de comunicação por troca de mensagens, quatro etapas. A primeira delas, vista na Figura 4, é a etapa de busca de serviços. Nela, o comprador informa ao seu agente o tipo de compra que deseja realizar. O agente comprador, por sua vez, faz uma requisição ao Diretório de Serviços (também controlado por um agente) por agentes que registraram o serviço desejado. O Diretório de Serviços então retorna ao agente comprador uma lista de agentes vendedores que coincidem com a requisição.

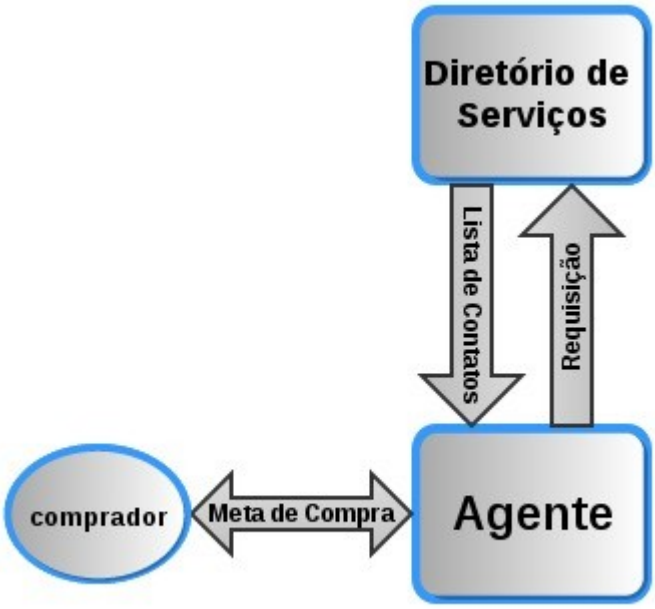

Figura **4** – Agente comprador buscando por agentes vendedores

Após receber a lista de agentes, o agente comprador inicia as negociações com os agentes vendedores (Figura 5). Nessa etapa, a decisão sobre qual agente deverá ser engajado na negociação pode obedecer alguns critérios (menor preço, qualidade do produto, etc.) pré-estabelecidos pelo usuário ou pelo agente.

Figura **5** – Agente comprador recebe ofertas dos agentes vendedores

Em seguida, a decisão é tomada e o agente comprador entra em negociação com o agente vendedor escolhido para então, estabelecer um contrato. Essa etapa é ilustrada na Figura 6. É possível visualizar a partir da segunda etapa (Figura 5) que tanto o comprador quanto o vendedor deixam de ter participação direta na negociação, deixando as decisões para serem tomadas de forma autônoma pelos agentes.

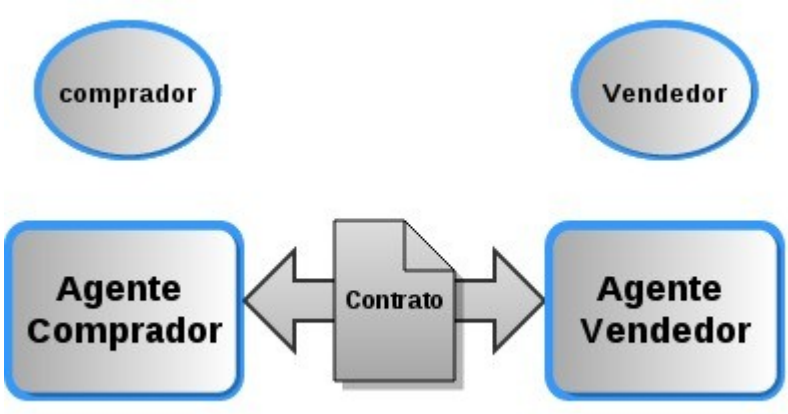

Figura **6** – Agentes estabelecem contrato para concluir a negociação

Com o contrato estabelecido, os agentes podem concluir a negociação. O agente comprador efetua o pagamento estabelecido no contrato e o agente vendedor entrega o produto desejado. A Figura 7 ilustra o desfecho desse processo de negociação, concluindo a demonstração de um sistema multi-agente de compra e venda onde os usuários exercem participação mínima, deixando todas as etapas da negociação a cargo dos agentes de software.

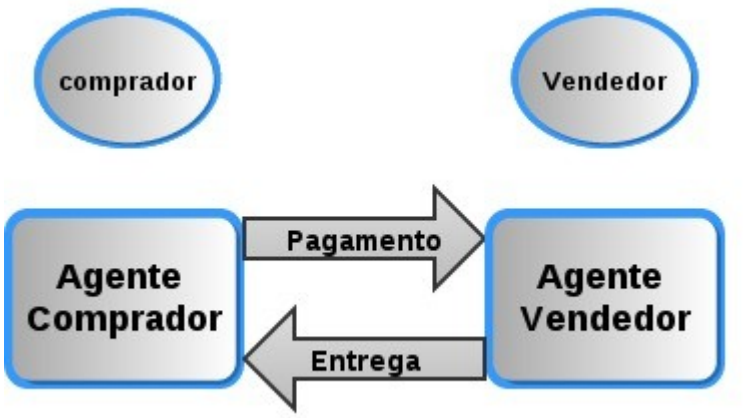

Figura **7** - Agentes concluem a negociação através do pagamento e entrega do produto

Em um sistema multi-agente, todo agente possui visão subjetiva, tendo informações incompletas do sistema devido às limitações de seu ponto de vista. Como consequência disso, cada agente tem seu próprio estado que não é acessível por outros participantes do sistema. Esses estados mudam individualmente de acordo com as regras de comportamento de seus agentes. As regras são influenciadas pelos dados pressentidos no ambiente que, por sua vez, é influenciado pelo agente atuando nele. Os dados são completamente descentralizados, distribuídos entre os agentes participantes e o ambiente.

Embora em certas ocasiões seja possível criar um sistema *single-agent* (com um único agente), um sistema multi-agente possui uma série de vantagens sobre sistemas *single-agent*:

- A robustez é um ponto forte, pois a falha de um agente pode ser superada caso outro agente tome conta da tarefa não concluída pelo primeiro;
- Em sistemas multi-agente, a escalabilidade é alcançada facilmente com a inclusão de novos agentes;
- Graças à modularidade, os agentes podem ser facilmente reintegrados em novos MAS.

Apesar dos comportamentos internos e externos os caracterizarem como um tipo único de sistema distribuído, sistemas multi-agentes precisam ser sustentados por uma genérica infraestrutura distribuída de computadores, ou um conjunto de

serviços *middleware*. Eles também necessitam de um ambiente de execução e um sistema de gerenciamento, incluindo segurança e armazenamento. Esses serviços middleware são fornecidos por uma Tecnologia de Informação e Comunicação (ICT) de um agente, também conhecido como plataforma MAS, na qual os agentes estão embarcados.

1.3 Arquitetura e Especificações FIPA

A Fundação para Agentes Físicos Inteligentes (FIPA) é uma organização internacional que desenvolve abertamente especificações que suportam a interoperabilidade entre agentes e sistemas baseados em agentes. Atualmente, os membros dessa fundação incluem profissionais (James Odell, Michael Miller, Anjanesh Babu, etc.), instituições (University of Otago, Institute for Systems and Tech. of Info., etc.) e corporações (Toshiba Corp., Siemens, Rockwell Automation, etc.). O objetivo da FIPA é promover a indústria de agentes inteligentes.

As especificações FIPA representam uma coleção de padrões com a intenção de promover a interoperação de agentes heterogêneos e serviços que eles podem representar (FIPA, 2014). Cada especificação é associada a um identificador assim que entra no ciclo de vida de especificações FIPA e é classificada de acordo com sua posição nesse ciclo. É possível visualizar essa classificação a partir do fluxograma mostrado na Figura 8.

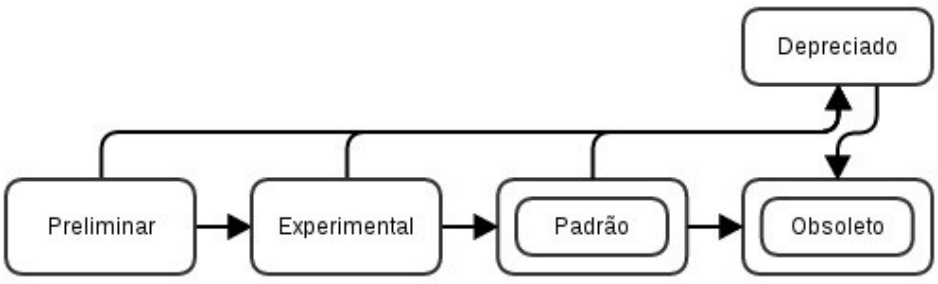

Figura 8 – Ciclo de uma especificação FIPA (FIPA, 2014)

O estado Preliminar é a concepção inicial da especificação, sendo proposto e gerado por Comitês Técnicos ou por Grupos de Trabalho supervisionados pelos Comitês. Por ser considerado um projeto em construção, uma especificação em estado Preliminar passa por várias revisões e mudanças, podendo inclusive ser rejeitada e considerada instável e inadequada para implementação.

Sob aprovação de um Comitê Técnico e da Mesa de Arquitetos, uma especificação é transferida para a fase Experimental, sendo considerada uma especificação estável e podendo ficar nesse estado por dois anos ou até ser promovida ao estado Padrão. Especificações em fase Experimental são consideradas adequadas para implementação em plataformas de agentes compatíveis com a FIPA. Se essa implementação for suficiente, a especificação passa para o estado Padrão.

Quando uma especificação atinge o estado Padrão, é considerada um padrão estável e formalmente publicado aprovado pelo corpo de padrões FIPA. Especificações que atingem o status Padrão são consideradas aptas para implementação em plataformas de agentes compatíveis com a FIPA. Qualquer modificação nessas especificações deve passar pela Mesa de Arquitetos FIPA. Não há avanço além do status Padrão, porém, uma especificação pode ser considerada Depreciada em qualquer fase do seu ciclo de vida.

Uma especificação classificada como Depreciada é potencialmente desnecessária para o padrão FIPA. Essa classificação pode ser dada devido a mudanças na tecnologia ou em outros padrões FIPA e pode ocorrer em qualquer fase do ciclo de vida (preliminar, experimental, padrão). Durante a fase de depreciação, a especificação é preparada para o estado final de obsolescência. Contudo, se não for aprovada para o estado Obsoleto, pode retornar ao ciclo de vida original, no estado em que estava antes da depreciação.

Para uma especificação atingir o estado Obsoleto, ela deve ser considerada desnecessária para o padrão FIPA. Essas especificações podem ser futuramente retiradas do estado Obsoleto, caso seja considerado necessário pela Mesa de Arquitetos FIPA e pelos membros FIPA. Caso isso ocorra, a especificação deve retornar ao estado Preliminar com um novo identificador.

Por mais de uma década, as atividades de padrões FIPA têm trabalhado para produzir especificações públicas para Sistemas Multi-Agentes, possibilitando suporte

à interoperabilidade, interação *open service* e suporte a desenvolvimento heterogêneo (Poslad, 2007).

A Figura 9 mostra a arquitetura abstrata especificada pela FIPA para a implementação de sistemas baseados em agentes. Nessa arquitetura, temos:

- *AMS* (Sistema de Gerenciamento de Agentes): um agente que fornece serviço de páginas brancas, ciclo de vida e serviço de gerenciamento de recursos;
- *DF* (Diretório Facilitador): um agente que fornece serviço de páginas amarelas;
- *Message Transport System* ou ACC (*Agent Communication Channel*): fornece serviço de comunicação para a troca de mensagens de agentes dentro e fora da plataforma.

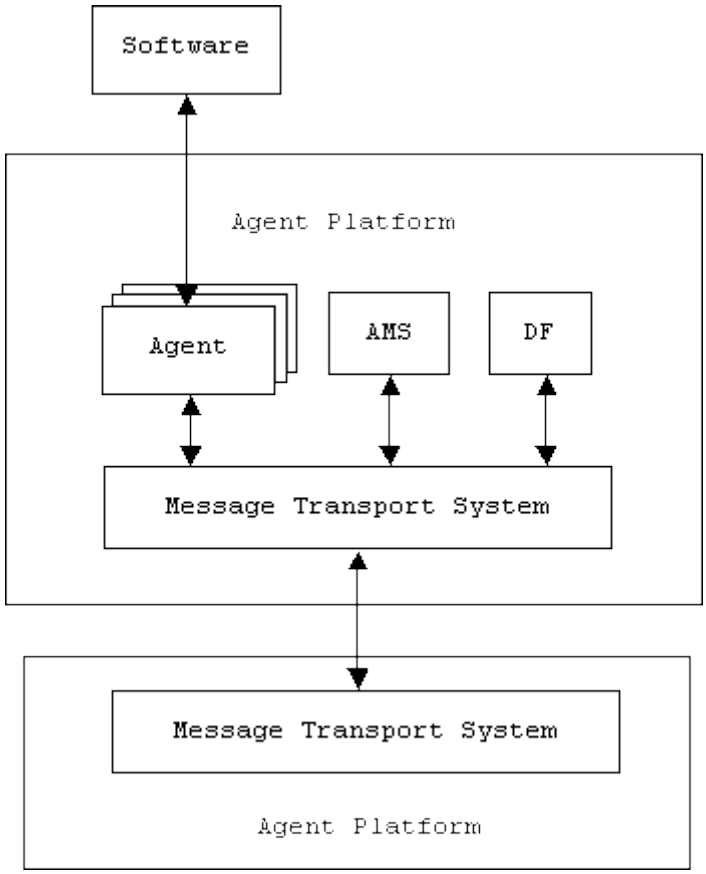

Figura 9 – Arquitetura Abstrata FIPA

O foco primário da Arquitetura Abstrata FIPA é criar a troca de mensagem semanticamente significativa entre agentes que poderão utilizar diferentes transportes de mensagens, diferentes linguagens para comunicação de agentes, ou diferentes conteúdos de linguagens (FIPA, 2002). A comunicação entre agentes está no núcleo do modelo FIPA para sistemas com agentes. Através da comunicação, agentes podem passar mensagens para outros agentes com o propósito de cumprir tarefas requeridas pela aplicação.

De acordo com as especificações FIPA, cada agente possui um nome, atributos e um localizador. Os agentes podem registrar e cancelar o registro em vários serviços de diretórios, além de poder procurar outros agentes em diferentes diretórios.

A comunicação entre agentes é feita utilizando a Linguagem de Comunicação de Agentes (ACL), através da passagem de mensagens assíncronas. As mensagens ACL podem ser definidas em vários tipos, dentre elas estão: *call for proposal*; *accept proposal*; *confirm*; *disconfirm*; *inform*; *inform if*; *request*; *propose*; *reject proposal* e *subscribe*. Além disso, é possível que os agentes entrem em acordo com um vocabulário em comum para facilitar a conversação. Esses vocabulários são chamados ontologias. O padrão FIPA para interoperabilidade entre agentes define o formato em que essas mensagens são transmitidas. Esse formato compreende um número de campos e em particular estão:

- O **remetente** da mensagem;
- A lista de **destinatários**;
- A intenção comunicativa (também chamada de **performativa**), indicando o que os remetentes pretendem alcançar (REQUEST caso o remetente deseja que o destinatário realize uma ação, INFORM se o remetente deseja que o destinatário esteja ciente de um fato, etc.);
- O **conteúdo** da mensagem, ou seja, a informação real inclusa na mensagem;
- A **linguagem** da mensagem, ou seja, a sintaxe usada para expressar o conteúdo. Ambos remetente e destinatário devem ser capazes de codificar/analisar expressões que estão de acordo com essa sintaxe para que a comunicação seja efetiva;

- A **ontologia**, ou seja, o vocabulário de símbolos usado no conteúdo e seus significados. Ambos remetente e destinatário devem atribuir o mesmo significado aos símbolos para que a comunicação seja efetiva.
- Alguns campos para controlar várias conversações concorrentes e para especificar limites de tempo para receber respostas, tais como **conversation-id**, **reply-with**, **reply-by**, **in-reply-to**.

Uma mensagem ACL no padrão FIPA deve conter um ou mais parâmetros. Quais parâmetros serão necessários para uma comunicação efetiva entre agentes é relativo à situação. O único parâmetro obrigatório em qualquer mensagem ACL é o *performative*, embora seja esperado que toda mensagem ACL também contenha *sender*, *contente* e *receiver* (FIPA, 2002). O conjunto completo dos parâmetros em uma mensagem ACL, sem relação com sua codificação e implementação, é mostrado na Tabela 2.

Parâmetro	Categoria do Parâmetro
performative	Tipo de Communicate Acts
sender	Participante na comunicação
receiver	Participante na comunicação
reply-to	Participante na comunicação
Content	Conteúdo da mensagem
language	Descrição do conteúdo
Encoding	Descrição do conteúdo
Ontology	Descrição do conteúdo
Protocol	Controle de conversação
conversation-id	Controle de conversação
reply-with	Controle de conversação
in-reply-to	Controle de conversação
reply-by	Controle de conversação

Tabela 2 – Parâmetros da Mensagem ACL FIPA

1.4 Web Semântica

Com a evolução da *World Wide Web* e da ampliação das possibilidades de utilização, é comum esperar hoje que qualquer pessoa tenha acesso à Web e esteja familiar com a ideia de uma rede de informações acessível em qualquer lugar, por qualquer pessoa. Porém, como as páginas Web possuem apenas informação léxica, tanto o conteúdo dessas páginas como o relacionamento entre elas é difícil de ser compreendido por entidades de software por encontrarem-se, em sua maioria, em linguagem natural (Freitas, 2003). Esse problema de falta de semântica acaba sendo refletido nas pessoas que, muitas vezes, possuem dados parciais à respeito de alguma informação e não podem utilizá-los para encontrar essa informação.

Para Breitman (2005 *apud* Pickler), a Web atual é denominada Web Sintática, onde computadores apenas apresentam informação ao passo que o processo de interpretação dessas informações fica a cargo dos seres humanos, devido à complexidade em avaliar, classificar e selecionar informações e conhecimentos de interesse. A Web Semântica surge então para contrapor esse ponto, buscando mecanismos que capturem o significado das páginas, criando um ambiente no qual os computadores possam interpretar dados e recursos provenientes de várias fontes.

> *A Web Semântica representa a evolução da Web atual. Enquanto a web tradicional foi desenvolvida para ser entendida apenas pelos usuários, a Web Semântica está sendo projetada para ser compreendida pelas máquinas, na forma de agentes computacionais, que são capazes de operar eficientemente sobre as informações, podendo entender seus significados. Dessa maneira, eles irão auxiliar os usuários em operações na Web.* (Dziekaniak & Kirinus, 2004)

Como uma extensão da Web atual, a Web Semântica permite que as informações sejam transmitidas dentro de um contexto ou significado bem definido, permitindo melhor interação humano-computador. O termo “semântica” é definido como “o estudo do significado de palavras, frases, símbolos, sinais e o que eles representam”, porém, Guiraud (1975) aponta três ordens principais de problemas semânticos:

1) A ordem dos problemas psicológicos, que relaciona os estados fisiológicos e psíquicos dos interlocutores nos processos de comunicação de signos;

2) A ordem dos problemas lógicos, que estabelece as relações dos signos com a realidade no processo de significação;
3) A ordem dos problemas linguísticos, que estabelece a natureza e as funções dos vários sistemas de signos.

O uso do termo “Web semântica” é baseado na segunda ordem, podendo ser justificado se observadas as possibilidades de associação de documentos a seus significados por meio de metadados descritivos, ou por meio de ontologias construídas e compartilhadas por desenvolvedores de aplicações. Sendo assim, o verdadeiro sentido da palavra “semântica” em Web Semântica é a de que autores de páginas tenham a capacidade de expressar aquilo que eles querem dizer na Web para que a máquina possa ler e utilizar essas informações.

Em 2001, o *World-Wide Web Consorcium* (W3C) apresentou uma proposta (Koivunen & Miller, 2001) definindo novas camas para a Web. Essas camadas podem ser vistas na Figura 10. A proposta inclui a padronização de linguagens para cada camada, começando pelas camadas Unicode e URI, que garantem a utilização de conjuntos de caracteres internacionais, fornecendo meios para identificação de recursos na Web Semântica.

A camada XML com definição de *namespace* e *schema* garante a integração da Web Semântica com outros padrões baseados em XML. Já a camada RDF e RDFSchema torna possível realizar afirmações e definir vocabulários que podem ser referidos por URIs. Essas camadas, juntamente com a camada de Ontologia, merecem destaque e serão abordadas mais adiante nesta seção.

A camada de ontologias é responsável por oferecer a expressividade necessária à representação de ontologias através da extensibilidade de RDFS para definir construções que implementam características de *frames* e *Description Logic* (Freitas, 2003).

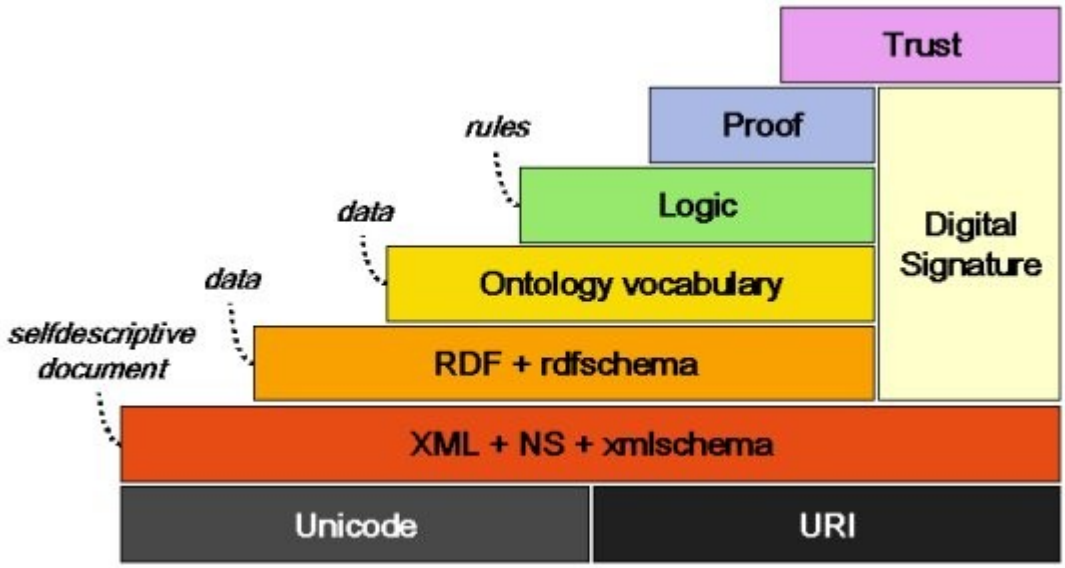

Figura 10 - Camadas da Web Semântica (Koivunen & Miller, 2001)

Para que os dados e documentos sejam associados a seus significados e transmitidos pela Web, os desenvolvedores contam com linguagens capazes de criar ontologias, descrever recursos e dados. Dentre essas linguagens, a eXtensible Markup Language (XML), Resource Desciption Framework (RDF) e Web Ontology Language (OWL) são consideradas fundamentais para o futuro da Web Semântica e recomendações W3C.

1.4.1 A linguagem XML

A partir das necessidades de descrever o conteúdo semântico e da estruturação dos documentos, criou-se a eXtensible Markup Language (XML), um formato de texto simples e flexível derivado do *Standard Generalized Markup Language* (SGML) que desempenha um papel importante na troca da larga variedade de dados na Web. Ele se tornou uma recomendação W3C em 1998 e hoje é a ferramenta mais comum para transporte de dados entre todos os tipos de aplicações.

De acordo com Dziekaniak e Kirinus (2004), a linguagem XML supre as deficiências da HTML ao permitir a criação de *tags* definidas pelo próprio usuário, proporcionando uma descrição mais detalhada dos recursos em termos de metadados.

Os autores ainda afirmam que, para que um documento XML seja considerado bem formado, devem-se respeitar duas regras: as *tags* devem ser aninhadas e os atributos devem ser únicos. Dessa forma, o documento segue uma estrutura em árvore, facilitando na recuperação das informações via Web.

Para Freitas (2003), a linguagem XML é uma abstração da HTML, sendo na verdade, uma *meta-linguagem* de editoração, por permitir a representação de outras linguagens de forma padronizada.

As principais funções do XML são: estruturar; transportar e armazenar dados. Os elementos dos documentos seguem uma estrutura em árvore, começando em um nó raiz, passando por galhos até as folhas. O elemento (conhecido como *tag*) deve ser definido pelo programador para representar o conteúdo desejado e cada elemento pode conter atributos e sub-elementos (filhos), formando uma hierarquia, onde elementos em um mesmo nível são chamados de irmãos. Para analisar e fornecer acesso ao conteúdo e estrutura de um documento XML, um módulo de software conhecido como processador XML é utilizado.

De acordo com Bray *et al* (2006), os objetivos projetados para o XML são:

1. XML deve ser diretamente utilizável através da Internet;
2. XML deve suportar uma grande variedade de aplicações;
3. XML deve ser compatível com SGML;
4. Deve ser fácil escrever programas que processem documentos XML;
5. O número de recursos opcionais em XML deve ser mantido absolutamente mínimo, idealmente zero.
6. Documentos XML devem ser legíveis para humanos e razoavelmente limpos;
7. O projeto XML deve ser preparado rapidamente;
8. O projeto XML deve ser formal e conciso;
9. Documentos XML devem ser fáceis de serem criados;
10. Concisão em marcações XML é de mínima importância.

A Figura 11 mostra o diagrama de um livro em uma livraria, representando elementos, seus atributos e a relação entre elementos de diferentes níveis. Já a Figura 12 mostra o documento XML desse livro.

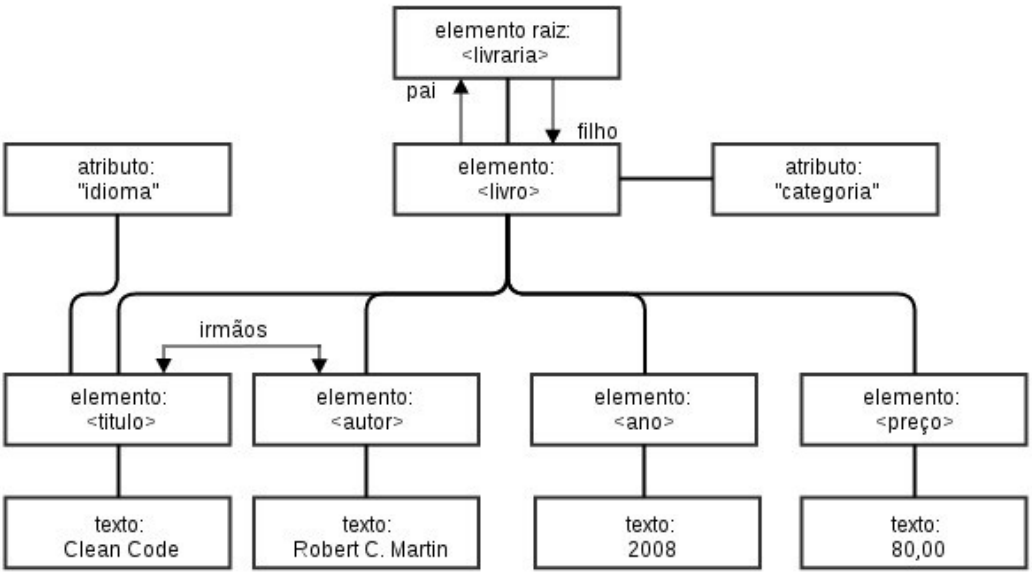

Figura 11 – Diagrama de um livro em uma livraria.

```
<livraria>
    <livro categoria="Eng_Software">
        <titulo idioma="ingles">Clean Code</titulo>
        <autor>Robert C. Martin</autor>
        <ano>2008</ano>
        <preco>80.00</preco>
    </livro>
</livraria>
```

Figura 12 – Documento XML da livraria representada na Figura 2

Algumas vantagens técnicas do XML são:

- É baseado em texto simples, ou seja, não é um formato binário;
- Suporta o padrão Unicode, permitindo a comunicação em linguagem humana;
- Pode representar estruturas de dados como árvores, listas e registros;
- Pode ser editado em qualquer ambiente, tais como editores de texto antigos e IDEs especializadas.

Os recursos disponibilizados pelo XML são tantos que desde a sua adoção pela comunidade, linguagens e frameworks foram criados a partir dele e, por serem baseadas em XML, são compatíveis entre si.

1.4.2 Resource Description Framework (RDF)

O RDF tem por objetivo definir um mecanismo de representação de metadados para descrever recursos não vinculados a um domínio específico de aplicação. Resultado do trabalho em conjunto desenvolvido por várias comunidades. (Dziekaniak & Kirinus, 2004)

A visão da Web Semântica é aumentar a Web Sintática para que recursos sejam mais facilmente interpretados por programas e agentes inteligentes. O desenvolvimento do RDF é uma tentativa de dar suporte à criação efetiva, troca e uso de anotações na Web (Pan, 2009).

Uma das principais linguagens baseadas em XML, o *Resource Description Framework* (RDF) é uma linguagem para representar informações na Web e, juntamente com o RDF Schema (RDFS) e a *Web Ontology Language* (OWL), é considerado a base para a Web Semântica. O RDF aborda um papel fundamental na Web Semântica: gerenciar dados distribuídos.

Na Web Semântica, informações são referidas como *recursos*. Um recurso é qualquer coisa que pode ser representada na Web, como o autor de um livro, o resultado de uma equação, ou até mesmo a Universidade Federal de Mato Grosso. Para expressar algo sobre esses recursos, o RDF equivale em termos formais às redes semânticas, onde os recursos são descritos em triplas (sujeito-predicado-objeto) (Freitas, 2003). Quando muitas triplas referem ao mesmo recurso, pode ser mais interessante visualizá-las em forma de *grafo direto*, no qual cada tripla é uma aresta de seu sujeito para seu objeto, com o predicado sendo o rótulo dessa aresta (Allemang, 2011). Essa forma de interpretação é visualizada na Figura 13.

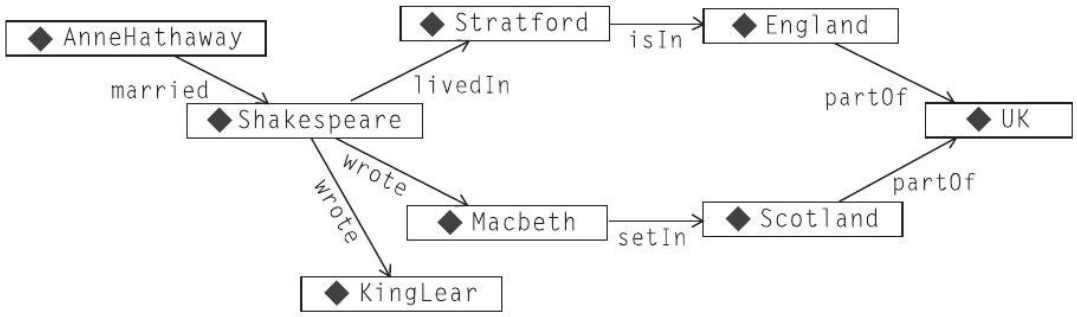

Figura 13 - Representação de triplas em forma de grafo (Allemang, 2011).

Assim, a partir do grafo gerado na figura 13, é possível construir triplas como:

AnneHathaway married Shakespeare
Shakespeare wrote Macbeth
Macbeth setIn Scotland
Scotland partOf UK

Fazendo uso de Identificadores Uniforme de Recursos (URIs), com RDF é possível identificar globalmente recursos na Web, criando conexões entre eles e até mesmo dizendo se dois recursos com URIs diferentes são equivalentes.

Uma vez que uma célula em uma tabela de um Banco de Dados é representada por três elementos (linha, coluna e valor), o bloco básico de construção para o RDF é chamado de tripla. Os elementos dessa tripla e suas relações com os valores de uma célula são: o *sujeito* da tripla equivale ao identificador da linha; o *predicado* da tripla equivale ao identificador da coluna; o *objeto* da tripla equivale ao valor da célula. Ao definir o sujeito *autor*, o predicato *escreceu* e o objeto *livro1*, é possível utilizar o método de serialização *Turtle* para escrever a seguinte tripla:

autor escreveu livro1 .

Mais adiante, pode-se definir predicados como *titulo*, *nome*, *ano_publicacao* e acrescentar informações sobre o autor e o livro:

autor nome “Robert C. Martin” .
livro1 titulo “Clean Code” .
livro1 ano_publicacao “2008” .

O RDF pode ser utilizado em diversas áreas, para diferentes propósitos. Na área de descoberta de recursos, o RDF possibilita a implementação de mecanismos de pesquisa mais eficientes. Na área de catalogação, é possível utilizá-lo para descrever recursos de informação de uma página Web, como em uma biblioteca digital. Já na área de agentes inteligentes, o RDF pode facilitar o compartilhamento de informações e conhecimento (Moura, 2001 citado por Dziekaniak & Kirinus).

Sendo um sistema de modelagem de dados, o RDF tem a flexibilidade como um de seus principais pontos forte. O relacionamento entre dois dados é representado de forma explicita, dispensando a necessidade de reorganizar tabelas para que elas igualem-se, ou se preocupar com dados faltantes em uma coluna em particular, pois um relacionamento (sujeito/predicado/objeto) ou está presente, ou não está. Com o uso dos URIs, é possível distribuir informações pela Web, unir informações de diferentes fontes e acrescentar informações sobre outras informações, de forma a dar suporte ao slogan *Anyone can say Anything about Any topic* (AAA).

1.4.3 Ontologias

A palavra *ontologia* é derivada do grego *ontos* e *logos* e significa, basicamente, “conhecimento do ser”. Breitman *apud* Pickler (2007) afirma que a palavra ontologia foi inserida no estudo da filosofia para distinguir o estudo do ser e o estudo dos vários tipos de seres vivos que existem no mundo natural, com o objetivo de fornecer sistemas de categorização para organizar a realidade. Para Moreira *et al* (2004), o termo “ontologia” tornou-se muito frequente em ciência da computação, sendo utilizado para denominar uma estrutura de termos e a relação entre eles em um determinado domínio. A autora aponta ainda que um dos principais

objetivos da utilização da ontologia na ciência da computação é construir bases de conhecimento interoperáveis e melhor estruturadas.

Para dar suporte à Web Semântica, a utilização de ontologias possibilita a descrição e distribuição dos dados com uma especificação formal e compreensível por máquinas em forma de:

- Terminologia, conceitos e nomenclaturas;
- Propriedades definindo conceitos;
- Relações entre conceitos;
- Regras distinguindo entre conceitos, definições, relações e restrições.

Com a grande quantidade de informações acumulando na Internet, o papel da ontologia para buscar, compartilhar e reusar essas informações é vital. Para Freitas (2003), a camada de ontologias é a mais importante para a Web Semântica, pois é responsável por oferecer a expressividade necessária para a representação de ontologias, sendo um passo importante na evolução da especificação do conhecimento.

Existem diversas formas de classificar os tipos e níveis de ontologias: quanto ao nível de atuação (conceitual ou simbólico); grau de formalismo (formal, informal, semi-formal) e abrangência (geral ou relacionada a um domínio específico). Contudo, nas áreas relacionadas à ciência da computação, existem divergências sobre como classificar essas ontologias. Isso pode ter sido causado pela falta de desenvolvimento de um embasamento teórico adequado para a terminologia na área (Moreira, 2004).

De maneira geral, em ciência da computação, uma ontologia é entendida como uma forma de representação de conhecimento sobre o mundo ou parte dele. Essas ontologias podem descrever indivíduos, classes, atributos e relacionamentos. A Figura 14 mostra um exemplo de ontologias utilizadas na linguagem OWL, uma linguagem criada à partir do RDF e utilizada para criar e instanciar ontologias na Web.

```
<owl:Class rdf:ID="Event"/>
<owl:Class rdf:ID="Album"/>
<owl:Class rdf:ID="Instrument"/>
<owl:Class rdf:ID="Musician"/>
<owl:Class rdf:ID="Admirer"/>
<owl:ObjectProperty rdf:ID="author">
        <owl:inverseOf>
                <owl:ObjectProperty rdf:ID="opus"/>
        </owl:inverseOf>
        <rdfs:domain rdf:resource="#Album"/>
        <rdfs:range rdf:resource="#Musician"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="player">
        <rdfs:range rdf:resource="#Musician"/>
        <rdfs:domain rdf:resource="#Instrument"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="loudness">
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="#Instrument"/>
</owl:ObjectProperty>
...
```

Figura 14 - Exemplo de ontologia em OWL

Dentre os benefícios advindos da utilização de ontologias, Freitas (2003) cita:

- A possibilidade para desenvolvedores reusar ontologias e bases de conhecimento. Como a construção de bases de conhecimentos equivale à tarefa mais cara e demorada em um projeto de sistemas especialistas e multi-agentes, a vantagem no reuso dessas bases é substancial;
- A possibilidade de tradução entre diversas linguagens e formalismos de representação de conhecimento. Essa tradução concretiza um ideal perseguido por gerações de pesquisadores da área de Inteligência Artificial. Isso permite o reuso do conhecimento e a comunicação entre agentes em formalismos diferentes.
- O acesso on-line a servidores de ontologias, capazes de armazenar milhares de classes e instâncias, servindo a várias empresas e grupos de pesquisa, podendo funcionar como ferramentas para manter a

integridade do conhecimento compartilhado, garantindo um vocabulário uniforme.

Derivada de conceitos utilizados em várias áreas do conhecimento, a ontologia constitui uma técnica imprescindível para as necessidades de comunicação e reuso automático de informações e conhecimento. Desde o desenvolvimento de sistemas multi-agentes, até a modelagem Orientada a Objetos, o uso de ontologias representa um grande passo para a representação do conhecimento e fundamental para a Web Semântica.

2. MATERIAS, TÉCNICAS E MÉTODOS

O processo de desenvolvimento da aplicação apresentada neste trabalho seguiu as seguintes etapas:

- Recebimento da proposta para criação de uma aplicação multi-agente para intercomunicação e acesso a ontologias;
- Estudos e análise de tecnologias disponíveis para o desenvolvimento da aplicação, como XML, RDF, OWL, JENA, JADE;
- Configuração do ambiente de trabalho para a implementação da aplicação;
- Modelagem e construção das ontologias a serem acessadas pelos agentes;
- Implementação dos agentes, estabelecimento da intercomunicação e acesso à ontologia.

Para a implementação e execução da aplicação, foi utilizado um notebook (pessoal) Dell XPS L701x com as seguintes configurações:

- CPU Intel Core i7-740QM;
- 16GB DDR3 1333MHZ (4x4GB);
- Tela LED 17.3”
- 1TB (2 x 500GB) Serial ATA (7200RPM);
- Placa Gráfica NVidia GeForce GT445M;
- Sistema Operacional Linux OpenSuse 13.1

Para a construção do ambiente de trabalho, os seguintes softwares foram instalados:

- Protégé Desktop 4.0 (Build 113);
- Eclipse Kepler;

- JADE 4.3.2;
- Java 1.7.0_51

A IDE Eclipse foi utilizada em conjunto com o Framework JADE para a implementação dos agentes. Já o software Protégé conta com um ambiente completo para a criação de ontologias. A seguir, maiores detalhes serão apresentados sobre as ferramentas utilizadas para a implementação da aplicação.

2.1 Framework JADE

JADE (*Java Agent DEvelopment Framework*) é uma framework gratúito, implementado em Java utilizado para a programação de sistemas multi-agentes. Ele obedece as especificações FIPA e possui uma série de ferramentas gráficas com suporte a fases de *debugging* e *deployment*.

Um sistema baseado em JADE pode ser distribuído em máquinas distintas, que executam sistemas operacionais distintos, pois são executados em JVMs. Os agentes podem até mesmo serem transferidos de uma máquina para outra em tempo de execução, sempre que for solicitado.

Dentre as vantagens do JADE, estão o suporte à abstração de agentes, comunicação de agentes ponto à ponto e sistema de páginas amarelas.

O framework JADE inclui uma biblioteca de protocolos de interação e comportamentos genéricos para os agentes, que precisam ser customizados para as necessidades específicas da aplicação a fim de criar as capacidades desses agentes.

2.2 Protégé Desktop

Protégé Desktop é um poderoso ambiente open-source grátis para desenvolvimento de ontologias utilizando a linguagem OWL. Ele oferece suporte à criação e edição de uma ou mais ontologias em um único ambiente de trabalho através de uma interface completamente customizável.

Ferramentas de visualização permitem navegar de forma interativa pelos relacionamentos entre as ontologias. É possível também executar operações de refatoração, incluindo *merging* entre ontologias, mover axiomas entre as ontologias e renomeação de múltiplas entidades.

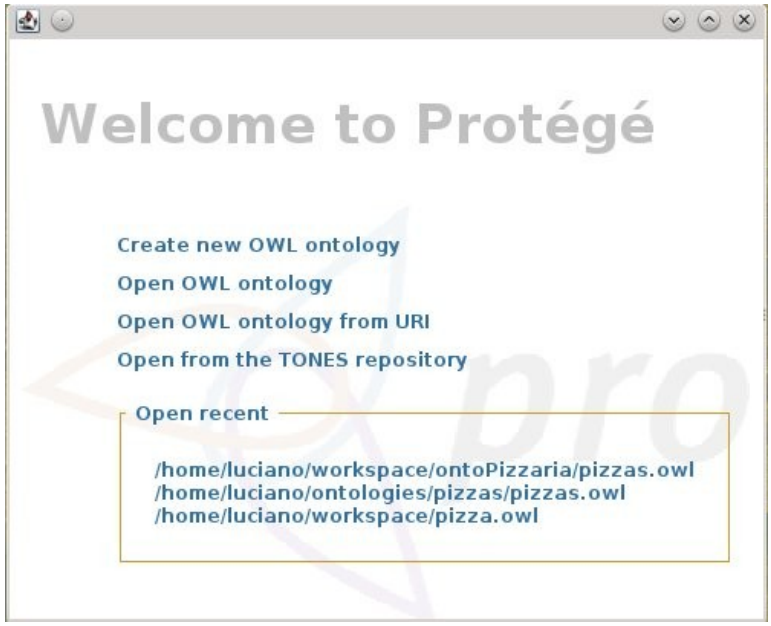

Figura 15 - Tela inicial da ferramenta Protégé 4.0

2.3 Linguagem OWL

Web Ontology Language (OWL) foi construída no topo do RDF (*Resource Description Framework*) e projetada para que aplicações que necessitam processar o conteúdo de informações, ao invés de apenas apresentá-las ao usuário. Com essa linguagem, pode-se representar de forma explícita termos em um vocabulário e o relacionamento entre eles, através da criação de ontologias.

OWL é hoje uma recomendação W3C e vista como uma tecnologia importante para o futuro da Web Semântica.

2.4 Ambiente de Desenvolvimento Eclipse

A plataforma Eclipse reúne um conjunto de ferramentas e uma IDE (Ambiente Integrado de Desenvolvimento) com suporte às linguagens de programação Java, PHP e C/C++, entre outras. Ela é distribuída de forma gratuita sob a licença EPL (Eclipse Public License) e segue o modelo Open Source, possuindo versões para Windows, Linux e Mac OS.

2.5 OWLAPI

OWLAPI é uma Interface de Programação de Aplicações (API) de código aberto, utilizada para criar, manipular e serializar ontologias OWL. Ela possui um reasoner interno, porém, para a execução deste trabalho foi utilizado o JFact, uma implementação em Java do reasoner FaCT++.

2.6 Etapas de desenvolvimento da aplicação

Na primeira etapa, foram realizadas reuniões com o Supervisor de Estágio, prof. Fernando Castilho, para a discussão sobre a necessidade da implementação de um sistema multi-agentes com acesso à ontologias para contribuição no projeto LAVI-França, realizado no Instituto de Computação da UFMT. Os detalhes e especificações da aplicação foram decididos e a etapa de estudos foi iniciada.

Na segunda etapa, foram realizados estudos no portal W3School sobre:

- *XML* (*eXtensible Markup Language*): designada para armazenar e transportar dados;
- *RDF* (*Resource Description Framework*): Linguagem para representar informações na Web através de ontologias;
- *RDFS* (*Resource Description Framework Schema*): conjunto de classes e propriedades utilizando a linguagem RDF;
- *OWL* (*Web Ontology Language*): Linguagem para representação do conhecimento construída no topo da linguagem RDF;
- *XML* Schema (*eXtensible Markup Language Schema*): especifica formalmente como descrever os elementos em XML;

- *XPath* (*XML Path Language*): linguagem utilizada para navegar através de documentos XML;
- *XQuery* (*XML Query Language*): linguagem de consulta utilizada para acessar e extrair dados de documentos XML;
- *DTD* (*Document Type Definition*): conjunto de regras que definem tags e suas propriedades que podem ser utilizadas em um documento XML.

Durante essa etapa, foi realizada também a leitura do livro Semantic Web for the Working Ontologist — Effective Modeling in RDF and OWL, Second Edition, de Dean Allemang e Jim Hendler. As leituras foram essenciais para a escolha do framework e da linguagem utilizada para a implementação das ontologias nas etapas subsequentes.

A terceira etapa é definida pela configuração do ambiente de trabalho, incluindo a escolha da plataforma para implementação da aplicação (Eclipse Kepler), software para modelagem, criação e edição de ontologias (Protégé Desktop 4.0) e o framework para implementação de sistemas multi-agentes (JADE 4.3.2). Cada ferramenta será descrita detalhadamente mais adiante nesta seção.

A quarta etapa é focada na modelagem e criação das ontologias. Para isso, foi utilizada a ferramenta de modelagem de ontologias Protégé Desktop 4.3.2, na qual foram realizadas adaptações na ontologia pré-existente pizza.owl. Essa ontologia é utilizada como fonte de estudos sobre o Protégé utilizando a linguagem OWL.

As edições e adaptações realizadas em pizza.owl foram salvas em uma nova ontologia, intitulada pizzas.owl e posteriormente exportadas para a linguagem Java. A figura 16 a hierarquia de classes dessa ontologia na ferramenta Protégé 4.0.

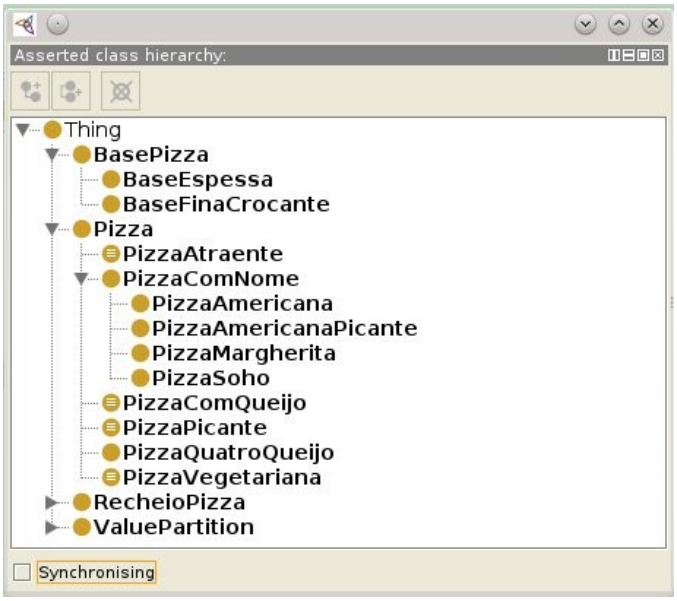

Figura 16 – Hierarquia de Classes da ontologia pizza.owl

Cada pizza é classificada de acordo as propriedades, base e recheios que ela pode ter. Uma mesma pizza pode pertencer a mais de uma superclasse, como é o caso da PizzaMargherita, mostrada na Figura 17, que se torna subclasse das classes PizzaVegetariana e PizzaComQueijo após as inferências sobre a ontologia serem realizadas.

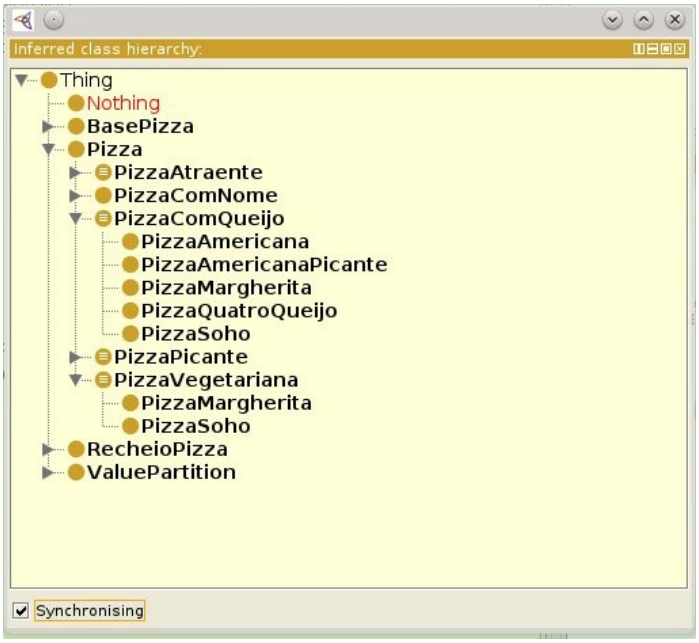

Figura 17 – PizzaMargherita é, ao mesmo tempo, uma PizzaComQueijo e uma PizzaVegetariana

A relação entre as bases, recheios e as classes de pizzas são expressas através de propriedades (predicados), na forma de *Pizza possuiRecheio Recheio*. Essas propriedades podem ser vistas na figura 18.

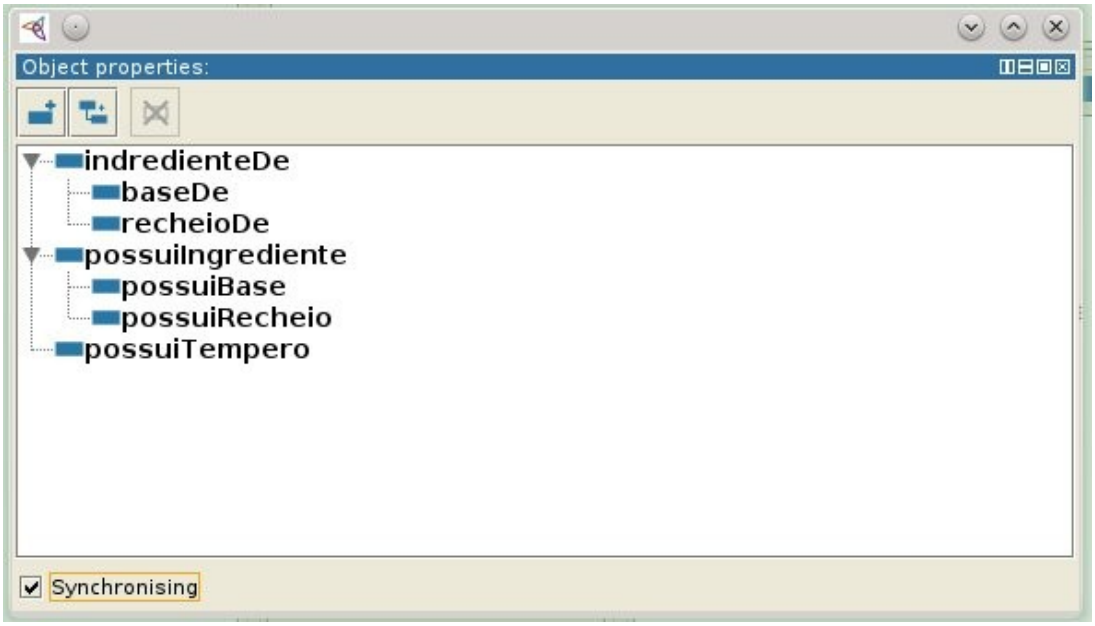

Figura 18 – Propriedades (predicados) da ontologia pizza.owl

Na quinta e última etapa, o agente Garçom e os agentes clientes foram implementados com o auxílio da IDE Eclipse Kepler, utilizando o framework JADE 4.3.2. A implementação dos agentes e os métodos de comunicação e acesso à ontologia serão descritos de forma detalhada a seguir.

2.7 Implementação e comunicação entre Agentes

Em um sistema baseado em agentes, as ações de cada agente são descritas em forma de *Behaviours* (comportamentos). Cada *Behaviour* possui um conjunto de uma ou mais ações que serão executadas de acordo com o contexto da aplicação. A comunicação entre agentes é feita através da troca de mensagens assíncronas do tipo ACLMessage. Nesse método, cada agente possui uma caixa de mensagens (pilha de mensagens) onde o sistema deposita as mensagens enviadas por outros agentes. Sempre que uma mensagem é depositada na pilha de mensagens, o agente

destinatário é notificado. O que o agente fará com essa mensagem (se vai retirar da pilha, quando vai retirar da pilha) fica a total decisão do programador.

A aplicação é composta por oito agentes principais: quatro agentes Clientes (Luciano, Nelcileno, Fernando e Paulo); um agente Garçom e um agente Pizzaria. Os outros dois agentes (DF e AMS) fazem parte da estrutura do JADE e são vitais para a execução de qualquer aplicação que utiliza esse framework. O agente Pizzaria possui como única função criar, ao início da execução, os contêineres Salão e Cozinha, para onde os agentes clientes e Garçom serão remanejados. O trecho de código com os métodos para criação de contêineres é mostrado na Figura 19. Para movimentar de um *container* para outro, um agente deve chamar o método **doMove()**, passando como parâmetro o *ContainerID* de destino.

```
private void criaContainerCozinha() {
    Profile cozinhaContainer = new ProfileImpl();
    cozinhaContainer.setParameter(Profile.CONTAINER_NAME, "Cozinha");
    this.runtime.createAgentContainer(cozinhaContainer);
}

private void criaContainerSalao() {
    Profile salaoContainer = new ProfileImpl();
    salaoContainer.setParameter(Profile.CONTAINER_NAME, "Salao");
    this.runtime.createAgentContainer(salaoContainer);
}
```

Figura 19 – Métodos do Agente Pizzaria para criação dos contêineres Salão e Cozinha

Os agentes clientes são subclasses da classe ClienteAgent. Essa classe contém os métodos de **registraServico()** e **moveParaPizzaria()**, responsáveis por registrar o serviço de cada agente Cliente no DF e movê-los para o contêiner Salão. Dessa forma, cada agente Cliente fica responsável apenas pela comunicação com o agente Garçom. Os *Behaviours* desses clientes incluem: pedido de pizza; pedido de cardápio e pedido de sugestões sobre qual pizza escolher. Esses pedidos são definidos pela ontologia passada ao agente Garçom. Um pedido de pizza possui a ontologia “pedido-pizza”, enquanto que o pedido do cardápio possui a ontologia “pedido-cardapio” e o pedido de sugestões é feito com a ontologia “pedido-sugestao”. A Figura 20 mostra o método **realizaPedido()** do agente Cliente Luciano, no qual o Behaviour para realização de pedidos é criado.

```
private void realizaPedido() {
    addBehaviour(new RealizaPedido("pedido-pizza", "Margherita"));
}
```

Figura 20 – Behaviour com um pedido de Pizza Margherita do Agente Cliente Luciano

Ao iniciarem, cada Agente é movido para o container Salão, criado pelo agente Pizzaria e, em seguida, registra seu serviço no *Directory Facilitator* (DF), também conhecido como “serviço de páginas amarelas” e aguarda o agente Garçom enviar uma mensagem solicitando um pedido.

O agente Garçom é movido para o Salão ao ser iniciado e então registra seu serviço no DF. Após isso, ele procura nas páginas amarelas por clientes que aguardam atendimento. O agente Garçom envia uma solicitação de pedido para cada cliente e, após receber os pedidos, move para o container Cozinha, criado pelo agente Pizzaria, simulando um pedido de preparação de pizzas. O agente então retorna ao Salão e entrega os pedidos, na forma de Identificadores Uniformes de Recursos (URIs) de cada pizza, para seus respectivos agentes.

Para que os pedidos (URIs) sejam entregues aos seus respectivos destinatários (agentes Clientes), foi utilizado um *HashMap* que armazena, antes do agente Garçom mover para a *container* Cozinha, o nome do agente Cliente e o pedido realizado. Quando o agente Garçom retorna ao *container* Salão, ele faz uma consulta a esse *HashMap* para realizar as entregas.

Como um agente é notificado sempre que uma mensagem é recebida, o agente Garçom realiza um atendimento (inicia uma conversação) assim que um agente Cliente realiza um pedido. Contudo, enquanto que pedidos de pizza são armazenados em um *HashMap* para que sejam atendidos após o Garçom voltar do *container* Cozinha, pedidos de sugestão/cardápio não requerem a movimentação do agente Garçom e, portanto, são atendidos de imediato.

A Figura 21 mostra um fluxograma que representa o comportamento do Garçom durante a comunicação com os agentes Clientes.

Os agentes DF e AMS são iniciados automaticamente assim que o contêiner principal é criado (*Main Container*). O AMS (*Agent Management System*) assegura que cada Agente possui um nome único e representa a autoridade na plataforma, sendo capaz de criar e matar agentes quando requisitado. O DF é responsável pelo

fornecimento do serviço de Páginas Amarelas, por onde um agente pode localizar outros agentes que oferecem serviços necessários para atingir seus objetivos.

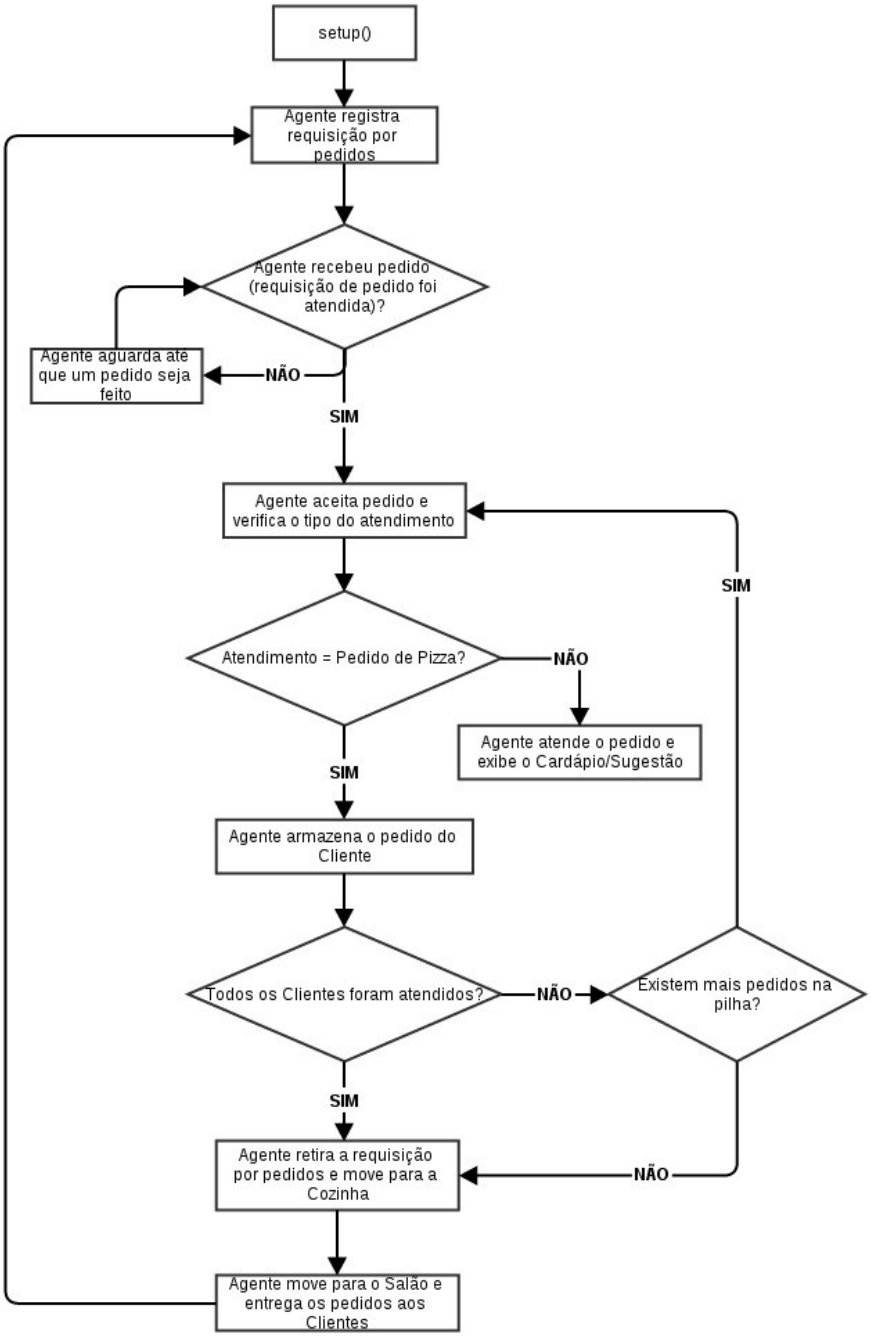

Figura 21 – Fluxograma da comunicação entre Garçom e Clientes

3. RESULTADOS

Com a execução das atividades propostas no estágio, o resultado obtido foi uma aplicação multi-agente que realiza acesso a uma ontologia e compartilha informações retiradas dela. Essa aplicação exemplifica o potencial e flexibilidade de sistemas multi-agentes quando unidos às tecnologias que englobam a Web Semântica. Sendo assim, o projeto de pesquisa LAVI-França obtém uma aplicação que poderá ser utilizada como fonte de estudos e aprendizado para futuras etapas do projeto.

O nome proposto para a aplicação é OntoPizzaria. Ao ser executada, a OntoPizzaria imprime no console as etapas da comunicação entre os clientes e o garçom. É possível observar que, por conta do método assíncrono de troca de mensagens (ACLMessage), quando o agente Garçom atende um agente Cliente, outros Agentes Clientes também podem ser atendidos até que a conversa com o primeiro seja encerrada. Esse assincronismo pode ser observado na Figura 22.

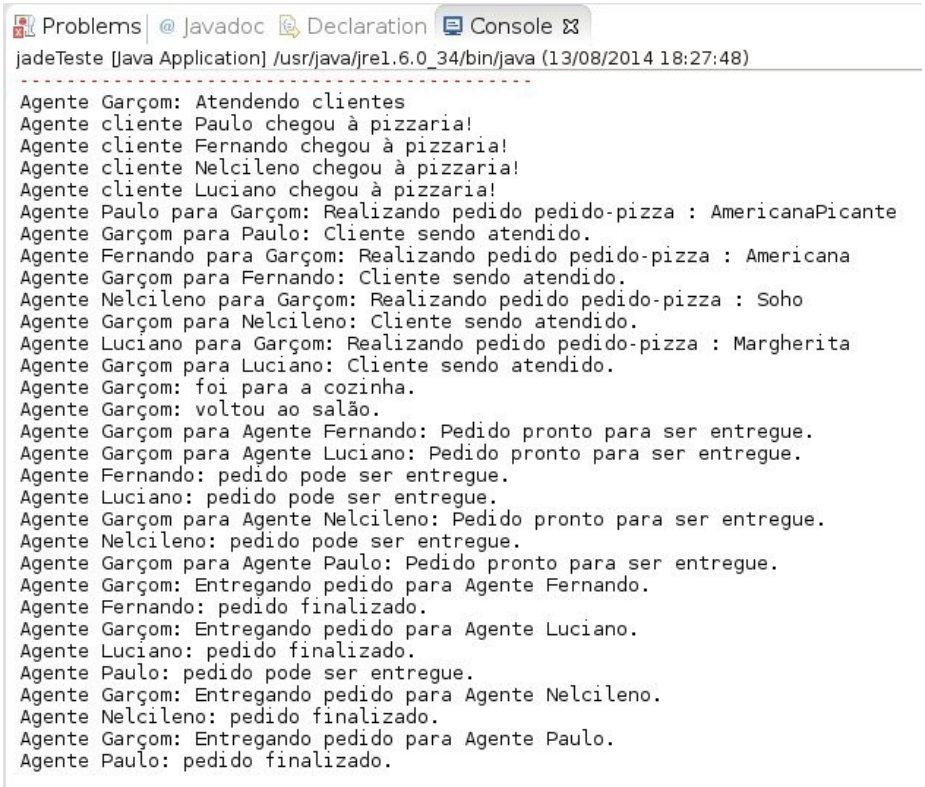

Figura 22 – Execução da aplicação. Comunicação sendo feita de forma assíncrona

É possível selecionar quais agentes serão iniciados na aplicação. Dessa forma, pode-se verificar o comportamento individual de cada agente ao realizar o pedido. Como exemplo, tem-se a figura 23, onde o agente Luciano realiza o pedido de uma PizzaMargherita. Já na figura 24, o agente Nelcileno pede por sugestões de pizzas com recheio de Parmesão. O agente Garçom então retorna a PizzaSoho, única pizza que possui esse recheio.

```
Agente cliente Paulo chegou ao Salão!
Agente cliente Fernando chegou ao Salão!
Agente cliente Nelcileno chegou ao Salão!
Agente cliente Luciano chegou ao Salão!
Agente Garçom chegou ao Salão!
Agente Garçom: Atendendo clientes
Agente Luciano para Garçom: Realizando pedido-pizza : Margherita
Agente Garçom para Luciano: Cliente sendo atendido.
Agente Garçom chegou à Cozinha!
Agente Garçom chegou ao Salão!
Agente Garçom para Agente Luciano: Pedido pronto para ser entregue.
Agente Luciano para Agente Garçom: pedido pode ser entregue.
Agente Garçom: Entregando pedido para Agente Luciano.
Agente Luciano: Recebido http://www.pizza.com/ontologias/pizzas.owl#Margherita
pedido finalizado.
```

Figura 23 – Agente Luciano realizando pedido de PizzaMargherita

```
Agente cliente Nelcileno chegou ao Salão!
Agente cliente Paulo chegou ao Salão!
Agente cliente Luciano chegou ao Salão!
Agente cliente Fernando chegou ao Salão!
Agente Garçom chegou ao Salão!
Agente Nelcileno para Garçom: Realizando pedido-sugestao : RecheioParmesao
Agente Garçom: Atendendo clientes
Agente Garçom para Nelcileno: Cliente sendo atendido.
Agente Garcom para Agente Nelcileno: Nossas opções de RecheioParmesao são:
http://www.pizza.com/ontologias/pizzas.owl#PizzaSoho;
```

Figura 24 – Agente Nelcileno pedido sugestões de pizzas com recheio Parmesão

Para demonstrar parte do potencial do *reasoner*, o agente Fernando foi programado para pedir sugestões de pizzas vegetarianas. Essa demonstração é possível porque a classe PizzaVegetariana fica vazia enquanto as inferências não são realizadas. Uma vez que o *reasoner* é acionado e as classificações são feitas, todas as subclasses de PizzaComNome que possuem somente recheios do tipo RecheioQueijo e/ou RecheioVegetal são automaticamente classificadas como PizzaVegetariana. O resultado disso pode ser visto na figura 25, onde o agente Garçom entrega ao agente Fernando as opções de PizzaVegetariana.

```
Agente Garçom chegou ao Salão!
Agente Garçom: Atendendo clientes
Agente cliente Fernando chegou ao Salão!
Agente Fernando para Garçom: Realizando pedido-sugestao : PizzaVegetariana
Agente Garçom para Fernando: Cliente sendo atendido.
Agente cliente Nelcileno chegou ao Salão!
Agente cliente Paulo chegou ao Salão!
Agente cliente Luciano chegou ao Salão!
Agente Garcom para Agente Fernando: Nossas opções de PizzaVegetariana são:
http://www.pizza.com/ontologias/pizzas.owl#PizzaSoho;
http://www.pizza.com/ontologias/pizzas.owl#PizzaMargherita;
```

Figura 25 – Agente Fernando pedindo sugestões de PizzaVegetariana

Devido à flexibilidade dos agentes e de seus comportamentos, é possível expandir a aplicação através de novas funcionalidades, como o cadastro de novos sabores e recheios de pizzas, inclusão de novas propriedades de modo que o *reasoner* faça mais inferências, registro das variedades de pizzas e recheios no DF para que os próprios agentes Clientes criem suas pizzas ou até mesmo a comunicação remota entre agentes clientes e garçom em diferentes máquinas.

4. DIFICULDADES ENCONTRADAS

Sistemas multi-agentes, Web Semântica e ontologias são temas ainda não explorados nas disciplinas do curso de Ciência da Computação da UFMT. Sendo assim, foram necessários muitos estudos e pesquisas sobre esses temas para que a aplicação pudesse ser desenvolvida.

Outras dificuldades encontradas na realização do estágio foram obtidas nas etapas de criação da ontologia e desenvolvimento da aplicação.

No início da criação da ontologia pizzas.owl, a versão utilizada do Protégé Desktop foi a 5.0. Porém, essa versão encontra-se na fase beta e apresenta problemas de retrocompatibilidade, gerando incompatibilidade com a ontologia original pizza.owl. Para solucionar esse problema, foi utilizado o Protégé Desktop 4.0, eliminando qualquer falha apresentada anteriormente.

Na etapa de desenvolvimento da aplicação, foi utilizada a API OWLAPI para o acesso e manipulação da ontologia. Porém, o *reasoner* padrão dessa API

apresentou problemas no momento de fazer as inferências, não retornando resultados. Esse problema foi solucionado com a utilização de um outro *reasoner*, chamado JFact. As inferências então foram efetuadas e o Agente Garçom pôde acessar todos os recursos necessários.

Outros problemas na etapa de desenvolvimento envolvem a especificação do protocolo MTP utilizado na comunicação entre os Agentes, a inicialização de múltiplos agentes em uma mesma aplicação e a criação de múltiplos contêineres em uma mesma aplicação. Para a criação dos contêineres, foi criado o Agente Pizzaria, responsável exclusivamente por essa função. Os problemas com o protocolo MTP e a inicialização de múltiplos agentes foram solucionados através da passagem de argumentos para a execução da aplicação. O argumento para a especificação do protocolo foi:

-mtp "jade.mtp.http.MessageTransportProtocol(http://10.10.2.65:7778)"

Já o argumento para a inicialização de múltiplos agentes foi:

"Fernando:cliente.FernandoAgent;Nelcileno:cliente.NelcilenoAgent;Luciano:cliente.LucianoAgent;Paulo:cliente.PauloAgent;Garcom:garcom.GarcomAgent;Pizzaria:pizzaria.PizzariaAgent"

Para a solução de todos os problemas citados, foram necessárias várias pesquisas na Internet sobre os erros gerados durante a execução da aplicação.

5. CONCLUSÕES

Com a experiência e o aprendizado obtidos durante a realização do estágio, nota-se as grandes possibilidades ao unificar a capacidade e autonomia na tomada de decisões de agentes de software com os conceitos que envolvem a Web Semântica, mais especificamente o acesso à ontologias e compartilhamento de recursos.

Com a aplicação desenvolvida, o projeto LAVI-França adquire uma fonte para estudos e pesquisa. A ontologia pizzas.owl pode receber novas propriedades e classes, gerando novas relações e recursos para serem exploradas pelos agentes. A flexibilidade dos agentes permite que eles sejam utilizados até mesmo em outras

ontologias (Locadora de Filmes, Classificação de Animais e Plantas, etc.), necessitando de poucas alterações em seu código-fonte.

Sistemas multi-agentes e Web Semântica são temas explorados desde a década de 90 e é possível encontrar uma grande gama de materiais e exemplos na Internet. Ainda assim, existem diversas subáreas que necessitam de pesquisas, cabendo a institutos e pesquisadores explorarem ainda mais esses temas, promovendo a evolução da interação humano-computador.

Pode-se concluir então que, ao utilizar os conceitos disciplinares adquiridos durante o curso de ciência da computação e aplicá-los na criação de uma aplicação dentro de um projeto de pesquisa, são obtidos excelentes resultados na formação acadêmica do aluno, gerando ganhos tanto para o futuro profissional do aluno quanto para a continuidade do projeto de pesquisa e seus membros.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ALLEMANG, Dean; 2011. *Semantic Web for the Working Ontologist: effective modelling in RDFS and OWL*. 2ª Edição. Massachussetts : Morgan Kaufmann Publishers.

FIPA. Welcome to FIPA!. Disponível em www.fipa.org (acessado em 04 de agosto de 2014).

H. S. NWANA. Disponível em http://link.springer.com/chapter/10.1007/978-3-662-03678-5_2#page-1 (acessado em 04 de agosto de 2014).

HORRIDGE, Mathew; 2011. *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools*. Edição 1.3. Machester : The University of Manchester.

JENNINGS, Nicholas R.; 1999. Agent-Oriented Software Engineering. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL AND ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS (12 : 31 de Maio – 3 de Junho, 1999 : Cairo, Egito). *Multiple Aproaches to Intelligent Systems (Proceedings)*. Berlin : Springer-Verlag. 4-10.

SOUZA, Renato Rocha; 2004. A Web Semântica e suas contribuições para a ciência da informação. Ci. Inf., Brasília, v. 33, n. 1 (jan./abril 2004), p. 132-141.

San José State University. FIPA. Disponível em http://www.cs.sjsu.edu/~pearce/modules/lectures/abs/fipa.htm (acessado em 04 de agosto de 2014).

W3C. Introduction. Disponível em http://www.w3.org/XML/#intro (acessado em 06 de agosto de 2014).

W3Schools. XML Tutorial. Disponível em http://www.w3schools.com/xml/ (acessado em 03 de agosto de 2014).

Yoav Shoham. An Overview of Agent-Oriented Programming. Disponível em http://www.infor.uva.es/~cllamas/MAS/AOP-Shoham.pdf (acessado em 31 de julho de 2014).

ODELL, James. Agent Technology: An Overview. 2010. Disponível em: <http://www.jamesodell.com/Agent_Technology-An_Overview.pdf>. (acessado em: 11 de agosto de 2014).

CSC CATALYST. Ontology White Papel. 2011. Disponível em: <http://www.jamesodell.com/Ontology_White_Paper_2011-07-15.pdf>. (acessado em: 11 de agosto de 2014).

CARDOSO, Amílcar. Multi-Agent Systems. 2006. Disponível em: <http://eden.dei.uc.pt/~amilcar/ftp/ATDS-MAS.pdf>. (acessado em: 10 de agosto de 2014).

HOEK, Wiebe van Der; WOOLDRIDGE, Michael. Multi-Agent Systems. 2007. Disponível em: <http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/kr-handbook.pdf2>. (acessado em: 12 de agosto de 2014).

PANAIT, Liviu; LUKE, Sean. Cooperative Multi-Agent Learning: The State of Art. 2004. Disponível em: <http://cs.gmu.edu/~eclab/papers/panait05cooperative.pdf>. (acessado em: 10 de agosto de 2014).

FREITAS, F. . Ontologias e a Web Semântica. In: Renata Vieira; Fernando Osório. (Org.). Anais do XXIII Congresso da Sociedade Brasileira de Computação. Volume

8: Jornada de Mini-Cursos em Inteligência Artificial. Campinas: SBC, 2003, v. 8, p. 1-52.

PAN, Jeff Z.. Resource Description Framework. 2009. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-540-92673-3_3#page-1>. (acessado em: 13 de agosto de 2014).

BRAY, Tim et al. Extensible Markup Language (XML): 1.1 (Second Edition). 2006. Disponível em: <http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf>. (acessado em: 14 de agosto de 2014).

JENNINGS, Nicholas R.. Artificial Intelligence: On agent-based software engineering. 2000. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0004370299001071>. (acessado em: 11 de agosto de 2014).

DIGNUM, Frank; DIGNUM, Virginia; DASTANI, Mehdi. Software Agents: Theory and Practice. Disponível em: <http://www.staff.science.uu.nl/~dignu101/agents.pdf>. (acessado em: 11 de agosto de 2014).

CHOPRA, Amit K.; SINGH, Munindar P.. An Architecture for Multiagent Systems: An Approach Based on Commitments. Disponível em: <http://www.csc.ncsu.edu/faculty/mpsingh/papers/mas/aamas-promas-09.pdf>. (acessado em: 14 de agosto de 2014).

MOREIRA, Alexandra; ALVARENGA, Lídia; OLIVEIRA, Alcione de Paiva. O nível do conhecimento e os instrumentos de representação: tesauros e ontologias. 2004. Disponível em: <http://www.dgz.org.br/dez04/Art_01.htm>. (acessado em: 10 de agosto de 2014).

SCHUMACHER, M.. Objective Coordination in Multi-Agent System Engineering. 2001. Disponível em: <http://link.springer.com/chapter/10.1007/3-540-44933-7_2#page-1>. (acessado em: 10 de agosto de 2014).

AZARIA, A.; AUMANN, Y.; KRAUS, S.. Autonomous Agents and Multi-Agent Systems. 2013. Disponível em: <http://link.springer.com/journal/10458#page-1>. (acessado em: 12 de agosto de 2014).