**UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ**
***CAMPUS* LUIZ MENEGHEL**

**THIAGO DE ATAÍDE ORLANDINI**

# USO DE *CHATTERBOTS* PARA GERAÇÃO DE LINGUAGEM NATURAL COMO RETORNO DE UM SISTEMA

Bandeirantes
2012

**THIAGO DE ATAÍDE ORLANDINI**

# USO DE *CHATTERBOTS* PARA GERAÇÃO DE LINGUAGEM NATURAL COMO RETORNO DE UM SISTEMA

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Glauco Carlos Silva

Bandeirantes

2012

**THIAGO DE ATAÍDE ORLANDINI**

# USO DE *CHATTERBOTS* PARA GERAÇÃO DE LINGUAGEM NATURAL COMO RETORNO DE UM SISTEMA

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

**COMISSÃO EXAMINADORA**

_____

Prof. Me. Glauco Carlos Silva
Faculdades Luiz Meneghel

_____

Prof. Me. Ederson Marcos Sgarbi
Faculdades Luiz Meneghel

_____

Prof. Me. Ricardo Gonçalves Coelho
Faculdades Luiz Meneghel

Bandeirantes, __ de_____ de 2012

Dedico este trabalho à minha mãe Maria Cecília, sem a qual eu não teria tido forças de realizá-lo.

## AGRADECIMENTOS

Agradeço à Deus por ter dado o dom da sabedoria e assim ter conseguido realizar este trabalho.

Agradeço a minha família, ao meu pai Claudio pelo apoio dado durante todos estes anos de estudo, a minha mãe Maria Cecília por me esperar acordada até tarde da noite durante todos estes anos de curso.

Agradeço aos meus amigos, aos companheiros de van que tornaram o caminho até a faculdade menos cansativo, aos colegas de sala que estiveram presentes nas brincadeiras, nas poucas partidas de futsal nos horários vagos e nos momentos de dificuldade do curso.

Agradeço ao meu orientador Glauco Carlos Silva, pela dedicação e ajuda por esse ano de trabalho.

Agradeço aos professores do curso de Sistema de Informação do *Campus* Luiz Meneghel – UENP, pelos ensinamentos passados, por auxiliar nas dúvidas e por momentos de distrações, sem eles não seria capaz de realizar este trabalho.

Agradeço a todos que não pude mencionar o nome, mas também foram importantes durante todo este processo de evolução pessoal por qual passei.

*Não há derrota que*
*derrote quem nasceu*
*para vencer.*
*(Autor Desconhecido)*

## RESUMO

Com a evolução da tecnologia e o uso do computador por pessoas foi necessário desenvolver sistemas que podem se comunicar com usuários e retornar na forma de linguagem natural as respostas às suas perguntas sobre um determinado assunto. Com o surgimento do *chatterbot*, essa interação humano-computador tornou-se possível. Através do Processamento de Linguagem Natural os *chatterbots* realizam uma pesquisa em seu banco de dados em *Artificial Intelligence Markup Language* e retornam para o usuário a resposta de sua pergunta. A fim de aplicar uma maior expressão ao *chatterbot*, um estudo foi realizado em *Web Ontology Language* e como integrar chatterbot e ontologias. Foi realizada uma forma de tratar e organizar informações usando uma abordagem baseada em *Artificial Intelligence Markup Language*. Como principais atividades, temos a criação de um *chatterbot* denominado Bot partindo de um conjunto de códigos em *Artificial Intelligence Markup Language*, usado como um sistema de atendimento online de uma pizzaria e integrando a uma base de dados em *Web Ontology Language*.



**Palavras-Chave:** Chatterbot, AIML, Processamento de Linguagem Natural, Ontologias, OWL.

## ABSTRACT

*With the evolution of technology and the use of computers by people was necessary to develop systems that can communicate with users and return in the form of natural language answers to your questions about a particular subject. With the emergence of chatterbot, this human-computer interaction became possible. Through natural language processing chatterbots perform a search in their database in Artificial Intelligence Markup Language, and return the user to the answer of your question. In order to apply a greater expression to the chatterbot, a study was conducted in Web Ontology Language and how to integrate chatterbot and ontologies. We performed a way to treat and organize information using an approach based on Artificial Intelligence Markup Language. As main activities, we are creating a chatterbot named robot based on a set of codes in Artificial Intelligence Markup Language, used as an answering system online a pizzeria and integrating a database into Web Ontology Language.*



***Keywords:*** *Chatterbot, AIML, Natural Language Processing, Ontologies, OWL.*

## LISTA DE QUADROS

## LISTA DE FIGURAS

## LISTA DE ABREVIATURAS E SIGLAS

AIML - *Artificial Intelligence Markup Language*

PLN - Processamento de Linguagem Natural

XML - *eXtensible Markup Language*

OWL - *Web Ontology Language*

RDF - *Resource Description Framework*

RDF-S - *Resource Description Framework Schema*

W3C - *World Wide Web Consortium*

FAQs - *Frequently Asked Questions*

# SUMÁRIO

# 1 INTRODUÇÃO

Com o passar dos anos a necessidade de comunicação entre humanos e máquinas se tornou cada vez mais importante. A interação entre ambos se torna cada vez mais essencial e a necessidade da utilização de linguagem natural seja interpretada por computadores passou a ser um fator fundamental nessa interação.

Com o surgimento dos *chatterbots*, robô com a capacidade de conversar em forma de chat, a linha que divide a comunicação entre humanos e máquinas ficou mais estreita. Hoje em dia são utilizados em diversas áreas, tais como: ensino a distância, comércio eletrônico, sites de buscas, entre outras atividades.

Entre os *chatterbots* se destacam o ELIZA que foi desenvolvido para se comportar como um terapeuta com intuito de fazer os pacientes responderem perguntas revelando informações pessoais. Na qual, a construção das frases consiste em inverter algumas frases ditas pelo paciente transformando-as em perguntas (Canuto, 2005). E o ALICE, um *chatterbot* desenvolvido em AIML (*Artificial Intelligence Markup Language*). Vencedor o prêmio Loebner em 2000, 2001 e 2004, sendo o primeiro a receber uma nota dos juízes maior que a de um ser humano (MOURA, 2008).

Os *chatterbots* simulam uma conversa por troca de mensagens, para que possam simular a conversa de forma convincente, utilizam a técnica do Processamento de Linguagem Natural (PLN) que tem como objetivo interpretar e gerar textos em linguagem natural (MOURA, 2008). O conhecimento sobre um determinado domínio de conhecimento é representado por objetos em AIML (*Artificial Inteligence Markup Language*), que é uma linguagem derivada do XML (*eXtensible Markup Language*). Os objetos AIML são compostos de categorias e tópicos que possuem dados interpretados ou não (MERENIUK, 2004).

Entretanto a AIML pode ser limitada para representar o conhecimento para tanto e para adicionar uma maior expressividade na linguagem XML utiliza-se a ontologia Owl (*Web Ontology Language*), a Owl facilita maior interpretabilidade do seu conteúdo suportando Xml e RDF (*Resource Description Framework)*, RDF

Schema e (RDF-S), fornecendo vocabulário adicional juntamente com uma semântica formal (*World Wide Web Consortium, 2004*).

Contudo, há muito no que evoluir em relação a interação e comunicação homem/máquina. Existe todo um processo para que o computador consiga entender e interpretar as dúvidas geradas em linguagem natural pelo homem e possa o mesmo responder.

## 1.1 CONTEXTO DO TRABALHO

A evolução da informática exigiu que a comunicação entre computador e homem pudesse ser trabalhada de forma mais fácil, na qual o computador consiga interpretar com clareza e facilidade aquilo que o usuário está dizendo e/ou perguntando através da linguagem natural, e assim realizar uma busca em sua base de conhecimentos e retornar a melhor resposta cabível.

Aplicando a técnica da Inteligência Artificial do Processamento de Linguagem Natural os *chatterbots* conseguem interpretar e gerar textos em linguagem natural e a partir destes textos simularem conversas de forma convincente (MOURA, 2008).

As conversas dos *chatterbots* são interpretadas utilizando arquivos em linguagem AIML, chamados de objetos AIML. Os objetos AIML são compostos de categorias e tópicos, no qual contém códigos interpretados ou não. Assim como os *chatterbots*, os objetos AIML funcionam baseados no Processamento de Linguagem Natural (MERENIUK, 2004)

## 1.2 FORMULAÇÃO DO PROBLEMA

A necessidade de um sistema capaz de entender a linguagem natural utilizada pelos usuários em pesquisas e retornar uma resposta de acordo com aquilo que foi perguntado.

## 1.3 OBJETIVOS

Este trabalho tem como objetivo desenvolver um protótipo de sistema capaz de gerar respostas em linguagem natural com base em um retorno de um sistema a partir do uso de *chatterbots* utilizando diálogos em AIML integrados com ontologias OWL.

## 1.4 OBJETIVOS ESPECÍFICOS

- Selecionar ferramentas de chatterbot que possam ser utilizadas;
- Desenvolver um protótipo de *chatterbot, capaz* de retornar informações através de linguagem natural em um determinado domínio;
- Integrar um *chatterbot* AIML com ontologias OWL;
- Testar e avaliar o protótipo e o seu funcionamento.

## 1.5 JUSTIFICATIVA

Este trabalho justifica-se pela necessidade do desenvolvimento de um sistema capaz de entender o uso da linguagem natural e retornar uma resposta de acordo com aquilo que foi questionado. Integrando o AIML dos *chatterbots* com ontologias em OWL melhora-se o tempo de busca de uma informação, pois as informações contidas na ontologia estão dispostas em forma de árvore e assim a busca é realizada através de suas ramificações e a mesma busca sendo realizada em um objeto AIML seria realizada bloco a bloco. Dessa forma, há uma maior facilidade para o registro e busca imediata de informações em uma base de dados e eliminar os conflitos de comunicação entre usuário e sistema, pois ambos terão acesso aos mesmos termos, proporcionando muitas vantagens, entre elas podemos citar: um diálogo em linguagem natural, precisão das informações e nas versões *online*, estar disponível em tempo integral, acesso a distância.

De acordo com Lopes (2002), nas bases de dados bibliográficas, os campos de busca em que se pode pesquisar usando apenas termos e/ou conceitos da

linguagem natural são os títulos e resumos. No processo de classificação das palavras, as remanescentes são usadas para gerar índices que podem ser pesquisadas por linguagem natural, assim pode-se rastrear temas e seus respectivos conceitos terminológicos que podem ser encontrados nas listas de cabeçalhos de assunto, nas tabelas de classificações especializadas, listas de descritores, códigos taxonômicos, nomenclaturas e outros.

## 1.6 METODOLOGIA

Este trabalho é baseado no desenvolvimento e uso de *chatterbots* para interação entre humanos e computadores. Analisando o desenvolvimento de um protótipo de sistema capaz de interpretar a linguagem natural utilizado por usuários e retornar a resposta mais cabível de acordo com o que foi perguntado. Baseado na forma de trabalhar com a linguagem AIML e ontologia OWL no processo de linguagem natural, será desenvolvido um protótipo que acessara uma base de dados interpretando a requisição feita pelo usuário e após encontrar as informações o sistema retornará ao usuário através de linguagem natural.

## 1.7 ORGANIZAÇÃO

O trabalho está organizado da seguinte forma. Seção 2 apresenta a fundamentação teórica, no qual são apresentados os *chatterbots*, a linguagem AIML e a ontologia OWL. Na Seção 3 apresenta a proposta deste trabalho, no qual descreve todo o processo de implementação de um *chatterbot*, o seu desenvolvimento e funcionamento. Na seção 4 apresenta a conclusão. E por último as referências bibliográficas e anexo.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 CHATTERBOT

O termo *chatterbot* é originado da junção de duas palavras em inglês, *chatter* que significa “aquele que conversa” e *bot* que é a forma abreviada de “*robot*”, robô. Portanto *chatterbot* é um programa que simula a conversa com um ser humano utilizando linguagem natural (RAMOS, 2008). O *chatterbot* é um sistema que facilita a interação homem – máquina, na qual ocorre em tempo real através da linguagem natural (MOURA, 2008).

De acordo com Leonhardt, (2003) apud Moura, (2008):

> Os *chatterbots* são agentes inteligentes desenvolvidos para simular uma conversa através de troca de mensagens de texto. Para simular uma conversação convincente, os *chatterbots* utilizam uma técnica de Inteligência Artificial chamada de Processamento de Linguagem Natural (PLN) que tem por objetivo interpretar e gerar textos em uma língua natural.

Segundo Ramos (2008), ELIZA é considerado o primeiro *chatterbot* a ser desenvolvido. ELIZA foi desenvolvido para se comportar como um terapeuta com intuito de fazer os pacientes a responderem perguntas revelando informações pessoais. Com uma arquitetura simples e sua programação baseada em técnicas de casamento de padrões, ELIZA através do método psicanalítico Rogeriano responde as perguntas repetindo as frases do paciente (CANUTO, 2005).

A construção das frases de ELIZA consiste em inverter algumas frases ditas pelo paciente transformando-as em perguntas. Por exemplo, se o usuário digita a frase: “Acho que ninguém gosta de mim.”, a resposta de ELIZA seria: “O que faz pensar que ninguém gosta de você?”. Entretanto, esse tipo de técnica não é totalmente eficiente, em alguns momentos ELIZA retorna frases que não fazem sentido, dificultando o dialogo (CANUTO, 2005).

Outro exemplo de *chatterbot* é o ALICE, um *chatterbot* desenvolvido em AIML (*Artificial Intelligence Markup Language*). ALICE venceu o prêmio Loebner em 2000, 2001 e 2004, sendo o primeiro a receber uma nota dos juízes maior que a de um ser humano. ALICE é composto por 41 mil elementos chamados de categoria, na

qual cada categoria é constituído de uma pergunta e resposta ou estimulo e resposta (MOURA, 2008).

Segundo Ferreira (2008), hoje os *chatterbots* são divididos em várias categorias, indo desde simples "psicólogos virtuais" até *bots* de uso empresarial como FAQs (*Frequently Asked Questions*). Os *bots* podem ser divididos em: *chatterbots, news bots, fun bots, academic bots, bot design, commerce bots, government bots, knowledge bots, search bots, shopping bots, stock bots* e *updates bots*, dentre outras categorias menores ou extremamente híbridas.

1. ***Chatterbots*** (também *chatter bots*, *chatter-bots* ou *chatbots*) – o caso estudado mais diretamente no âmbito deste artigo. São programas que trabalham com linguagem, desenvolvidos para simular conversas com seres humanos;
2. ***News bots*** – programas "jornalistas", capazes de criar jornais personalizados baseados em algumas informações do usuário, e também de fazerem "clipagem" de artigos em versões online de jornais reais;
3. ***Fun bots*** – programas de diversão, como personagens de realidade virtual, ou *chatters* cômicos;
4. ***Academic bots*** – *bots* de ensino à distância, simulando aulas virtuais;
5. ***Bot design*** – capazes de produzirem outros *bots*;
6. ***Commerce bots*** – ligados ao comércio na internet;
7. ***Government bots*** – buscam informações em sites governamentais;
8. ***Knowledge bots*** – reúnem num mesmo programa agentes inteligentes, agentes de informação, agentes de laboratório, *cyberagents*, agentes da web, e outras ferramentas inteligentes de busca;
9. ***Search bots*** – buscadores da web;
10. ***Shopping bots*** – comparam e fazem compras para o usuário;
11. ***Stock bots*** – monitoradores de estoque; e
12. ***Update bots*** – informam de atualizações em sites da internet.

Os *chatterbots* ELIZA e ALICE podem ser analisados segundo algumas características (Laureano, 1999 apud Ramos, 2008), tais como:

- Aquisição de conhecimento: determina a capacidade de aprender do *chatterbot* durante o dialogo com o usuário;
- Memória: refere-se à capacidade de lembrar os diálogos passados, memorizando nomes de pessoas e assuntos discutidos.
- Domínio: refere-se aos domínios sobre os quais o *chatterbot* é capaz de dialogar. Alguns não fazem restrição de domínio. Outros podem ser especialistas em determinados assuntos. Existem ainda os que permitem ao usuário escolher qual o domínio da conversa.
- Robustez: define a capacidade do *chatterbot* para responder a sentenças inesperadas feitas pelo usuário.
- Conhecimento: determina a capacidade do robô de falar sobre si próprio.

## 2.2 AIML

Ferreira (2008) descreve o uso o AIML da seguinte maneira:

> Os *chatterbots* são baseados em AIML (*Artificial Inteligence Markup Language*), uma linguagem de XML (*eXtensible Markup Language*), os *chatterbots* recebem perguntas em linguagem natural, consultam uma base de conhecimento e, em seguida, fornecem respostas a perguntas, novamente em linguagem natural, utilizando sistemas de PLN (Processamento de linguagem natural).

O AIML descreve uma classe de dados, chamados de objetos AIML. Os objetos AIML são compostos de categorias e tópicos que possuem dados interpretados ou não. Quando um usuário realiza uma pergunta ao sistema, ela é analisada dentro do objeto AIML ele retorna a resposta, caso contrário o *chatterbot* retorna como um dado não interpretado (MERENIUK, 2004).

Abaixo segue um exemplo de dado não interpretado:

**USUÁRIO:** Você conhece X?

**CHATTERBOT:** Eu não conheço X?

No caso exemplificado acima o dado não foi interpretado, no qual X pode ser qualquer palavra ou nome.

O AIML é composto por categorias representadas pela *tag* <*category*>, a categoria é formada por uma pergunta, uma resposta ou um contexto opcional. A pergunta ou estimulo é passada através da *tag* <*pattern*>, a resposta através da *tag* <*template*> e contexto opcional é utilizado pelas *tags* <*that*> ou <*topic*> (MERENIUK, 2004).

No AIML existem várias *tags* utilizadas para facilitar a conversa do *bot* fazendo com que seja mais próxima de uma conversa real, tendo a impressão de conversar com outra pessoa. As principais *tags* do AIML segundo (Walacce, 2005 apud Wantroba et al., 2008) são:

<aiml>: endereça o início e fim do documento.

<*category*>: denota uma unidade na base de conhecimento.

<*pattern*>: armazena o que um usuário pode dizer ou escrever para o *bot.*

<*template*>: contém a resposta para uma entrada do usuário.

Um arquivo AIML começa com o cabeçalho informando as versões do XML e AIML, o mesmo é dividido em outras *tags*. A *tag* <*category*>, categoria, é responsável por dividir os assuntos tratados dentro do objeto AIML. Dentro das categorias possui a *tag* <*pattern*>, padrões, nela fica armazenada a entrada de texto que o usuário digitará em uma conversa com o *bot*. E a *tag* <*template*>, modelo, fica registrado o que o bot responderia, case lhe perguntasse algo (WANTROBA et al., 2008).

```
<?xml version=”1.0” encoding=”ISO-8859-1”?>
<aiml version= “1.0.1” >
<category
        <pattern> TUDO BEM COM VOCÊ </pattern>
        <template> Estou bem. E você? </template>
</category>
</aiml>
```

Quadro 1 – Exemplo do uso da linguagem AIML

Outra *tag* muito utilizada é a:

<*that*>: esta *tag* é utilizada para dar continuidade à conversa entre usuário e *bot.*

Como explicado acima, a *tag* <*that*> possibilita que o *bot* continue a conversa com o usuário. O uso desta *tag* consiste em deixar uma segunda resposta ao bot caso o usuário faça outra pergunta. Sabendo que o *bot* armazena a última sentença respondida em uma variável chamada "*topic*", a *tag* <*that*> consulta essa variável e compara com o que está contido na *tag* (WANTROBA et al., 2008).

| **Código AIML** | **Conversa Humano/Bot** |
|---|---|
| <category><br><pattern> ESTOU BEM </pattern><br><that> Estou bem. E você? </that><br><template> Que bom. </template><br></category> | Humano: Como vai você?<br>Bot: Estou bem. E você?<br>Humano: Estou bem.<br>Bot: Que bom. |

Quadro 2 – Exemplo do uso da *tag* <that>

Outra situação que surge entre as conversas é uma frase que pode ser dita de vários modos diferentes. Para resolver este problema, utiliza-se a *tag* <*srai*>, que evita a redundância de informações, sendo responsável por redirecionar a resposta de uma determinada questão à categoria que a (WANTROBA et al., 2008).

A *tag* <srai> é utilizada na implementação de recursividade dentro do AIML. Todavia existe um desacordo com o significado do acrônimo, no qual "A.I", significa inteligência artificial, o "S.R", não possui uma definição concreta, podendo significar "*stimulus-response*", "*syntactic rewrite*", "*symbolic rediction*", "*simple recursion*", ou "*synonym resolution*" (MERENIUK, 2004). Veremos abaixo, cada uma das várias formas de aplicar o uso da *tag* <srai> detalhadamente:

1. *Symbolic reduction* (Redução simbólica): Reduz formas gramaticais complexas em formas mais simples.

2. *Divide and Conquer* (Dividir e Conquistar): Divide uma entrada em duas ou mais subpartes, combinado uma resposta para cada.
3. *Synonyms*: Mapeia diferentes maneiras de dizer a mesma coisa para a mesma resposta.
4. Corrigindo gramaticalmente.
5. Detectando palavras chaves em qualquer lugar da entrada.
6. *Conditionals* (Condicionais): Certas formas de ramificação podem ser implementadas com o <srai>.
7. Qualquer combinação de (1) – (6)

O perigo de usar <srai> é que permite que o programador crie laços infinitos. Todavia, mesmo que o uso da <srai> por programadores novatos ofereça riscos, a utilização da *tag* é muito mais simples que o uso de um bloco interativo estruturado por marcadores de controle que possam substituí-lo (MERENIUK, 2004).

No AIML pode-se realizar uma substituição de qualquer cadeia de caracteres fazendo uso do símbolo asterisco (*), dependendo do interpretador pode-se utilizar a *tag* <*star*>.

| **Código AIML** | **Conversa Humano/Bot** |
|---|---|
| <category><br><pattern> ME CHAMO * </pattern><br>< template><br>Prazer em conhecê-lo *!<br></template><br><category> | Humano: Me chamo Thiago.<br><br>Bot: Prazer em conhecê-lo Thiago! |

Quadro 3 – Exemplo do uso do asterisco (*).

O AIML possui *tags* para armazenamento de informações, como o nome do usuário para o uso futuro na conversa. Para alterar ou armazenar o valor de uma variável utiliza-se a *tag* <*set*> e para ler o valor de uma variável utiliza-se a *tag* <*get*>. Essa operação deve ser transparente ao usuário e para que isso ocorra é

utilizado a *tag* <*think*>, responsável por utilizar as operações internas do *bot* (WANTROBA et al., 2008).

| **Código AIML** | **Conversa Humano/Bot** |
|---|---|
| <category><br><pattern> ME CHAMO * </pattern><br>< template><br><think><br><set= “nome” >*</set> </think><br>Prazer em conhecê-lo. <get= “nome” ><br></template><br><category> | Humano: Me chamo Thiago.<br><br>Bot: Prazer em conhecê-lo Thiago. |

Quadro 4 – Exemplo de uso das *tags* <set> e <get>

Para realizar buscas aleatórias utiliza-se a *tag* <*random*>, ela busca uma expressão aleatoriamente dentro do conjunto de respostas especificas na sua implementação e retorna ao usuário (WANTROBA et al., 2008).

```
<category>
        <pattern> COMO VAI VOCE </pattern>
        <template>
                <random>
                        <li> Eu vou bem, obrigado. </li>
                                <li> Hoje estou bem. </li>
                        <li> Tenho estado bem ultimamente. </li>
                        <li> Tudo certo comigo. </li>
                </random>
        </template>
</category>
```

Quadro 5 – Exemplo do uso da *tag* <random>

Atualmente o AIML suporta duas maneiras de interface com outras linguagens e sistemas. A *tag* <*system*> e <*javascript*>. Enquanto a primeira executa qualquer

programa acessível via linha de comando pelo sistema operacional, e insere o resultado de sua execução na resposta, a segunda, permite o uso de scripts (MERENIUK, 2004).

## 2.3 OWL

A W3C (*World Wide Web Consortium*) descreve a OWL (*Web Ontology Language*), como:

> Uma linguagem projetada para uso de aplicações que precisam processar o conteúdo de informações em vez de apenas apresentar informações para os seres humanos. Owl facilita maior interpretabilidade do conteúdo da web do que o suportado por Xml, RDF (*Resource Description Framework)*, RDF Schema e (RDF-S), fornecendo vocabulário adicional juntamente com uma semântica formal.

A OWL é uma ontologia que pode descrever classes, suas respectivas propriedades e suas instancias, construindo relacionamentos entre essas informações gerando uma Web Semântica. De forma geral é uma melhoria do RDF e XML ampliando a expressividade e o poder da linguagem (SILVA, 2007).

A Web Semântica é uma visão pra o futuro da web em que a informação é dada significado explícito, tornando mais fácil para máquinas processarem automaticamente e integrar as informações disponíveis na web. A Web Semântica é baseada sobre a capacidade do XML por ser uma forma simples de representação dos dados (*World Wide Web Consortium, 2004*).

De acordo Pereira (2006), a OWL constitui um passo gigantesco para a representação do conhecimento na Web. De forma simples e eficiente equilibra as necessidades exigidas pela indústria e as restrições inerentes ao desenvolvimento de uma linguagem ontológica que incorpore os resultados das mais recentes investigações da área.

Segunda Pereira (2006), o *Web Ontology Working Group* desenvolveu seis documentos sobre a OWL que foram considerados a base que contribuiu e possibilitou a obtenção da Recomendação W3C por parte da W3C:

- **OWL *Web Ontology Language Overview*:** É um documento que apresenta, de forma geral, as características da linguagem OWL;
- **OWL *Web Ontology Language Guide*:** É um documento que apresenta, de forma mais detalhada, as características da linguagem OWL bem como alguns exemplos de utilização;
- **OWL *Web Ontology Language Reference*:** É um documento que serve como referencia para os detalhes de cada uma das características da linguagem OWL;
- **OWL *Web Ontology Language Semantics and Abstract Syntax*:** É um documento que apresenta a semântica da linguagem OWL e detalhes sobre a conversão de informação de OWL para RDF;
- **OWL *Web Ontology Language Test Cases*:** É um documento que apresenta vários casos de testes práticos efetuados durante a utilização da linguagem OWL;
- **OWL *Web Ontology Language Use Cases and Requirements*:** É um documento que apresenta exemplos práticos de utilização da linguagem OWL

A linguagem OWL baseia-se na utilização do conceito de URI, na sintaxe RDF/XML e na linguagem Esquema RDF para gerar ontologias com as seguintes características:

- Facilidade de serem manipuladas de forma distribuída através de diferentes sistemas;
- Versáteis às necessidades da Web;
- Compatíveis com as normalizações web referentes à acessibilidade e à internacionalidade;
- Utilização de uma especificação aberta a facilmente extensível.

OWL foi projetado para atender a necessidade de uma Linguagem de Ontologia para Web. OWL é parte da pilha crescente de recomendações W3C relacionadas à Web Semântica (*World Wide Web Consortium, 2004*).

- **XML:** fornece uma sintaxe de superfície para documentos estruturados, mas não impõe restrições semânticas sobre o significado desses documentos.
- **XML *Schema*:** é uma linguagem para restringir a estrutura de documentos XML e estende-se também com tipos de dados XML.
- **RDF:** é um *datamodel* para objetos ("recursos") e relações entre eles, fornece uma semântica simples, e estes *datamodels* pode ser representado em uma sintaxe XML.
- **RDF *Schema*:** é um vocabulário para descrever propriedades e classes de RDF recursos, com uma semântica para a generalização de hierarquias de tais propriedades e classes.
- **OWL:** adiciona mais vocabulário para descrever propriedades e classes: entre outros, as relações entre classes (de disjunção, por exemplo), cardinalidade (por exemplo, "exatamente um"), a igualdade, a mais rica de digitação das propriedades, características das propriedades (simetria, por exemplo), e classes enumeradas.

OWL fornece três sub-linguagens, projetadas para uso por comunidades específicas de implementadores e usuários (*World Wide Web Consortium, 2004*). Sendo elas:

- **OWL *Lite*:** suporta aqueles usuários que necessitam principalmente de uma hierarquia de classificação e restrições simples. Por exemplo, enquanto ele suporta restrições de cardinalidade, ela só permite valores de cardinalidade 0 ou 1.
- **OWL DL:** suportam aqueles usuários que querem o máximo de expressividade, mantendo completude computacional, todas as conclusões são garantidas para ser computáveis, e todas as computações terminarão em tempo finito.
- **OWL *Full*:** é destinada a usuários que querem máximo a expressividade e a liberdade sintática do RDF sem garantias computacionais. Por exemplo, em OWL *Full* uma classe pode ser tratada simultaneamente como uma coleção de indivíduos e como um indivíduo em seu próprio direito.

Do mesmo autor, cada uma destas sub-linguagens é uma extensão de sua predecessora, tanto em relação ao que pode ser expresso, como em relação ao que pode ser concluído. O seguinte conjunto de relações é verdadeiro, já seu inverso não é:

- Toda ontologia OWL *Lite* válida é uma ontologia OWL DL válida.
- Toda ontologia OWL DL válida é uma ontologia OWL *Full* válida.
- Toda conclusão OWL *Lite* válida é uma conclusão OWL DL válida.
- Toda conclusão OWL DL válida é uma conclusão OWL *Full* válida.

## 2.4 INTERPRETAÇÃO DE LINGUAGEM NATURAL

A Interpretação de Linguagem Natural pode ser utilizada em diversas aplicações, tal uso justifica a compreensão desta ampla área. Abaixo veremos algumas aplicações que utilizam dessa técnica de Inteligência Artificial.

- Consulta em Banco de Dados utilizando Linguagem Natural;
- Robô “Ed”;
- *ChatterBot* Doroty;
- Sete Zoom; e
- *ChatterBot* Elektra.

As aplicações que tratam de linguagem natural consistem basicamente em dois segmentos:

- Aplicações baseadas em textos; e
- Aplicações baseadas em diálogos.

As aplicações baseadas em texto são sistemas que procuram documentos específicos em uma base de dados, tradutores de documentos e sistemas que resumem textos. As aplicações baseadas em diálogos referem-se às interfaces de linguagem natural para banco de dados, os sistemas tutores e os sistemas que interpretam e respondem a comandos expressados em linguagem escrita ou falada (OLIVEIRA NETO; TONIN; PRIETCH, 2004).

De acordo com Oliveira Neto, Tonin e Prietch (2004), existem quatro categorias históricas de programas para interpretação de linguagem natural, sendo elas:

- Alguns programas que tinham por objetivo a geração de um número reduzido de resultados em domínios específicos. A simplicidade do processo permitia que muitos dos problemas da linguagem natural pudessem ser ignorados;
- Em alguns dos primeiros sistemas, era armazenada uma apresentação do texto, recomendando-se a engenhos de indexação para auxiliar a recuperação de determinadas palavras ou frases. Como o texto armazenado podia cobrir qualquer assunto, os sistemas não eram restritos a um determinado domínio. Todavia, esses sistemas eram semanticamente fracos e não tinham poderes dedutivos.
- Sistemas de lógica limitada que tinham por objetivo traduzir frases de entrada para a notação formal usada na base de dados. Aqui a intenção era permitir que se fizessem deduções a partir da informação mantida na base de dados, mesmo que somente alguns dos processos utilizados na conversação do dia-a-dia pudessem ser explorados;
- Os sistemas com base de conhecimento, que se utilizavam da informação sobre um assunto específico para compreender as frases de entrada. Tais programas, alguns dos quais constituíam sistemas especialistas, exibiam vários poderes dedutivos.

Com a crescente necessidade de aplicações da lingüística computacional fez ressaltar a carência de dados lingüísticos de dimensões reais, em particular, de léxicos e gramaticais de grande cobertura (OLIVEIRA NETO; TONIN; PRIETCH, 2004). O processamento de linguagem natural possui vários problemas, como a grande variação morfológica e sintática das unidades lexicais ou ambigüidade intrínseca da língua portuguesa. Para a solução dos problemas citados destacam-se três níveis de análise lingüística: morfológico, sintático e semântico (OLIVEIRA NETO; TONIN; PRIETCH, 2004).

> Um sistema computacional interpreta uma sentença em linguagem natural, através da analise de informações morfológicas (léxicas), sintáticas (regras gramaticais) e semânticas (significados), armazenadas em um

> dicionário, juntamente com as palavras que o sistema compreende (OLIVEIRA NETO; TONIN; PRIETCH, 2004 apud Silva e Lima, 2007).

## 2.5 BASE DE DADOS AIML

A base de dados AIML é composta inicialmente pelos arquivos de saudação e despedida, na qual serão responsáveis pelas conversas simples entre usuário e bot. Foi desenvolvida outra base dados que simula um atendente de uma pizzaria online.

No quadro pode-se observar um exemplo utilizado na codificação AIML implementada na base de dados do bot.

```
<!-- Formas de Pagamento -->
<category>
        <pattern>QUAIS AS FORMAS DE PAGAMENTO</pattern>
        <template>Por ser um sistema online a forma de pagamento é realizada via
cartão de crédito.
                Ou pode-se pagar com dinheiro no ato da entrega.</template>
</category>

<category>
        <pattern>* FORMAS DE PAGAMENTO</pattern>
        <template>
                <srai>QUAIS AS FORMAS DE PAGAMENTO</srai>
        </template>
</category>
```

Quadro 6 – Exemplo da codificação AIML

## 2.6 ONTOLOGIA

De acordo com Gruber (1993) apud Moura (2008), as ontologias são desenvolvidas em comunidades diferentes e utilizadas para domínios de interesse diversos, mas a maioria delas possui alguns componentes básicos. Segue, abaixo, as descrições destes componentes:

- **Classes:** grupos de elementos formados por um conjunto de atributos iguais. Representam a unidade principal de uma ontologia e forma os conceitos que definem um determinado objeto;
- **Relações:** responsáveis pelos relacionamentos semânticos entre os conceitos de um determinado domínio, formando, assim, a taxonomia;
- **Axiomas:** são regras declaradas sobre as relações que devem ser cumpridas pelos elementos. Possibilitam inferir conhecimento que formam a taxonomia da ontologia;
- **Instâncias:** representam um determinado objeto de um conceito, ou seja, são os próprios dados da ontologia.

Para garantir o desenvolvimento correto de ontologias alguns itens precisam ser considerados, (Gruber, 1993 apud Moura, 2008):

- **Clareza:** uma ontologia deve ter clareza e ser objetiva ao comunicar o significado dos termos definidos. As descrições devem ser completas e documentadas em linguagem natural, além de independentes do contexto social e computacional;
- **Coerência:** as inferências sobre a ontologia devem ser consistentes com as definições axiomáticas. Essa coerência deve ser aplicada para os conceitos de modo formal e também para os informais. É considerado incoerente uma ontologia que apresenta contradição entre um axioma e seu exemplo informal;
- **Extensibilidade:** possuir a capacidade de definir novos termos para utilizações especiais, baseado em um vocabulário existente, sem a necessidade de rever as definições existentes;
- **Compromisso mínino na implementação:** a conceitualização da ontologia deve ser especificada no nível de conhecimento, sem se restringir a uma determinada tecnologia de representação de conhecimento;

- **Compromisso ontológico mínimo:** preocupação em atender à intenção da atividade compartilhada do conhecimento. Devem existir poucas imposições sobre o domínio que está sendo desenvolvido.

## 2.7 *PROTÉGÉ*

É um editor de ontologias de *open source* e um *framework* de base de conhecimentos. A plataforma *Protégé* suporta duas maneiras principais de modelagem de ontologia através dos editores *Protégé-Frames* e *Protégé-OWL*. As ontologias do *Protégé* podem ser exportadas para uma variedade de formatos, incluindo RDF(s). OWL, e XML Schema.

O *Protégé* é baseado em Java, é extensível, e proporciona um ambiente *plug-and-play* que faz com que seja uma base flexível para prototipagem rápida e desenvolvimento de aplicações.

Segundo Moura (2008), desenvolver uma ontologia no *Protégé* consiste em seguir o seguinte passo: definir um esquema de classes (*class*), subclasses (*subclass*), propriedades e relações (*slots*), referente ao domínio que se deseja modelar.

Do mesmo autor, nessa ferramenta é possível visualizar a hierarquia de classes no formato de árvore, onde são exibidos seus relacionamentos e suas instâncias e também é possível criar um esquema de *query*, para consultas futuras.

## 2.8 OWL API

A OWL Api é uma api Java de implementação de referência para criar, manipular e serializar ontologias OWL. É open source e está disponível sobre as Licença Apache e LGPL. A api inclui os seguintes componentes:

- Uma API para OWL2 e uma eficiente implementação de referência na memória;
- Analisador e escritor RDF/XML;
- Analisador e escritor OWL/XML;

- Analisar de Sintaxe Funcional e escritor de OWL;
- Analisador e escritor *Turtle*;
- Analisador KRSS;
- Analisador formato de arquivo OBO; e
- Interface *Reasoner* para trabalhar com pensadores como a FaCT++*, HermiT, Pellet e Racer*.

# 3 DESENVOLVIMENTO

O sistema proposto teve o intuito de facilitar a comunicação entre homem e computador. O protótipo irá acessar uma base de dados que contém todas as informações do sistema, após encontrar os dados desejados retornará os mesmos ao usuário por meio de linguagem natural. Para solucionar este problema pretende-se agregar a linguagem natural com base nas respostas de um sistema. Para um melhor entendimento da idéia proposta, apresenta-se o seguinte exemplo.

**Usuário:** Quais sabores de pizza com cobertura X?

**Sistema:** Os sabores de pizza com cobertura X são calabresa, pepperoni e frango.

O (X) representa qual tipo de cobertura de pizza que o usuário deseja saber, neste caso o usuário quer saber quais os sabores de pizza com cobertura de carne, na qual o sistema retorna calabresa, pepperoni e frango. O sistema deve possuir a funcionalidade de receber a mensagem inserida pelo usuário e analisá-la semanticamente e após gerar uma base de dados, na qual contém os dados referentes à mensagem do usuário.

Baseado no exemplo acima será desenvolvido um protótipo capaz de interpretar os dados contidos na base de dados e retorná-los ao usuário por meio de linguagem natural.

O protótipo será desenvolvido seguindo as estruturas básicas de um *chatterbot*, no qual o mesmo consegue simular uma conversa entre duas pessoas. Dessa forma, todas as mensagens do sistema retornarão por meio de diálogos entre computador e usuário.

Para acessar os campos necessários na base de dados do sistema, pretende-se utilizar a ontologia OWL, esta que possui uma melhor interação e facilidade de implementação em bases desenvolvidas no padrão XML, que é a linguagem raiz do AIML. Com a ontologia OWL, será criado um padrão para obtenção das informações

geradas pelo sistema que possam ser acessadas pelo *chatterbot* e retorne ao usuário por meio de uma conversa.

A estrutura AIML utilizada seguirá a estrutura básica de qualquer outro *chatterbot*. Abaixo segue a lista de *tags* a ser utilizadas:

- *Category;*
- *Pattern;*
- *Template;*
- *System;*
- *Think;*
- *Set;* e
- *Get*.

Por meio destas *tags* será agregada a linguagem natural na resposta retornada do sistema.

## 3.1 BASE PIZZA.AIML

A base de dados AIML é dividida em 4 categorias, cada qual relacionada a um determinado assunto, sendo eles:

- Formas de Pagamento;
- Locais de Atendimento;
- Tempo de Entrega; e
- Sabores.

Abaixo é exibido o código do bloco AIML desenvolvido:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aiml version="1.0.1" xmlns="http://alicebot.org/2001/AIML-1.0.1"
    xmlns:html="http://www.w3.org/1999/xhtml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://alicebot.org/2001/AIML-1.0.1
http://aitools.org/aiml/schema/AIML.xsd">
<!-- Copyright (c) 2007 ALICE A.I. Foundation, Inc. -->

<!-- Formas de Pagamento -->
<category>
      <pattern>QUAIS AS FORMAS DE PAGAMENTO</pattern>
```

```
      <template>Por ser um sistema online a forma de pagamento é
realizada via cartão de crédito.
            Ou pode-se pagar com dinheiro no ato da
entrega.</template>
</category>

<category>
      <pattern>* FORMAS DE PAGAMENTO</pattern>
      <template>
            <srai>QUAIS AS FORMAS DE PAGAMENTO</srai>
      </template>
</category>

<!-- Locais de Atendimento -->
<category>
      <pattern>QUAIS OS LOCAIS DE ATENDIMENTO</pattern>
      <template>Atendemos somente via o site
www.pizzaonline.com.br</template>
</category>

<!-- Tempo de Entrega -->
<category>
      <pattern>QUAL O TEMPO DE ENTREGA</pattern>
      <template>Após realizado o pedido a pizza é entregue no máximo
em 30 minutos.</template>
</category>

<category>
      <pattern>* TEMPO DE ENTREGA</pattern>
      <template>
            <srai>QUAL O TEMPO DE ENTREGA</srai>
      </template>
</category>

<!-- Sabores -->
<category>
      <pattern>QUAIS OS SABORES DE PIZZA QUE POSSO
ESCOLHER</pattern>
      <template>
            As pizzas são divididas em quatro sabores, sendo eles:
Carne, Peixe, Queijo e Vegetal.
      </template>
</category>

<category>
      <pattern>* SABORES DE PIZZA *</pattern>
      <template>
            <srai>QUAIS OS SABORES DE PIZZA QUE POSSO
ESCOLHER</srai>
      </template>
</category>
```

```
<category>
	<pattern>QUAIS OS TIPOS DE PIZZA</pattern>
	<template>
		<srai>QUAIS OS SABORES DE PIZZA QUE POSSO
ESCOLHER</srai>
	</template>
</category>

<category>
	<pattern>* TIPOS DE PIZZA</pattern>
	<template>
		<srai>QUAIS OS SABORES DE PIZZA QUE POSSO
ESCOLHER</srai>
	</template>
</category>

<category>
	<pattern>EU QUERO SABER OS DE *</pattern>
	<template>
		<srai>
			<set name="sabores"><star></set>
			SYSTEM
		</srai>
	</template>
</category>

<category>
	<pattern>* SYSTEM</pattern>
	<template>
		<system>
			::sabor::<get name="sabores"></get>
		</system>
	</template>
</category>
</aiml>
```

Quadro 7 – Base pizza.aiml

No Quadro 7 as perguntas foram desenvolvidas para responder as dúvidas dos clientes em relação às formas de pagamento, qual o tempo para ser entregue a pizza, quais os locais de atendimento e auxiliar a fazer o pedido informando os sabores de pizzas.

Os códigos circulados são responsáveis por obter o sabor de pizza que o usuário deseja, salvar em uma variável e depois passar através da *tag* <*system*> para a classe OntologiaPizza (Anexo B) e realizar a busca na ontologia.

Na Figura 1 é demonstrado o funcionamento do diálogo do chatterbot com um usuário.

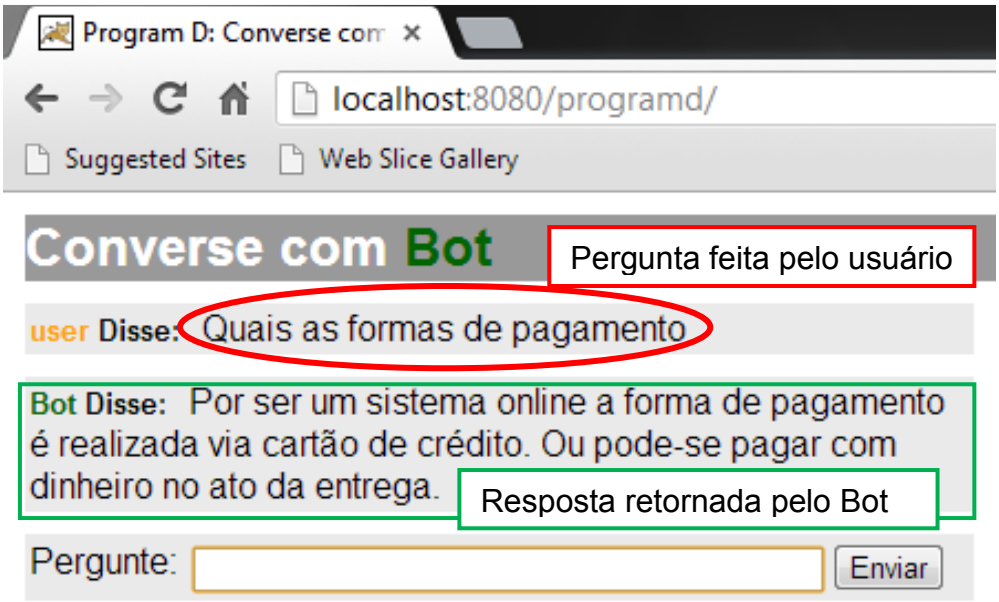


Figura 1 – Funcionamento do código pizza.aiml

No Quadro 8 encontra-se o código da *tag* <*system*>, na qual foi implementado para que receba o valor obtido no diálogo do *chatterbot* e repasse para a classe OntologiaPizza (Anexo B).

```
package org.aitools.programd.processor.aiml;

import org.w3c.dom.Element;
import org.aitools.programd.Core;
import org.aitools.programd.CoreSettings;
import org.aitools.programd.OntologiaPizza;
import org.aitools.programd.parser.TemplateParser;
import org.aitools.programd.processor.ProcessorException;

public class SystemProcessor extends AIMLProcessor{
    /** The label (as required by the registration scheme). */
    public static final String label = "system";

    /**
```

```
     * Creates a new SystemProcessor using the given Core.
     *
     * @param coreToUse the Core object to use
     */
    public SystemProcessor(Core coreToUse){
        super(coreToUse);
    }

    /**
     * @see AIMLProcessor#process(Element, TemplateParser)
     */
    @Override
    public String process(Element element, TemplateParser parser) throws
ProcessorException{

        String resposta = "Infelizmente não posso responder sua pergunta! Reformule-a
!!";

        CoreSettings coreSettings = parser.getCore().getSettings();

        // Don't use the system tag if not permitted.
        if (!coreSettings.osAccessAllowed())
        {
            logger.warn("Use of <system> prohibited!");
            return EMPTY_STRING;
        }

        String directoryPath = coreSettings.getSystemInterpreterDirectory().getPath();

        String prefix = coreSettings.getSystemInterpreterPrefix();

        String commandLine = parser.evaluate(element.getChildNodes());
```

```
        String[] comandoSplit = commandLine.split("::");

        String sabor = comandoSplit[2].trim();

        OntologiaPizza op = new OntologiaPizza();

        resposta = op.OntologiaPizza(sabor);

        System.out.println(resposta);

        return resposta;
    }
}
```

Quadro 8 – Implementação do Código da Tag <system>

## 3.2 ONTOLOGIA PIZZA.OWL

Com uso da ferramenta *Prótegé* foi desenvolvido a ontologia, na qual contém os sabores das coberturas das pizzas. As coberturas foram divididas em 04 categorias, sendo elas:

- Cobertura de Queijo;
- Cobertura de Carne;
- Cobertura de Vegetal; e
- Cobertura de Peixe.

Na Figura 2, vê-se a estrutura da ontologia pizza, na qual tem a classe CoberturaPizza e suas ramificações.

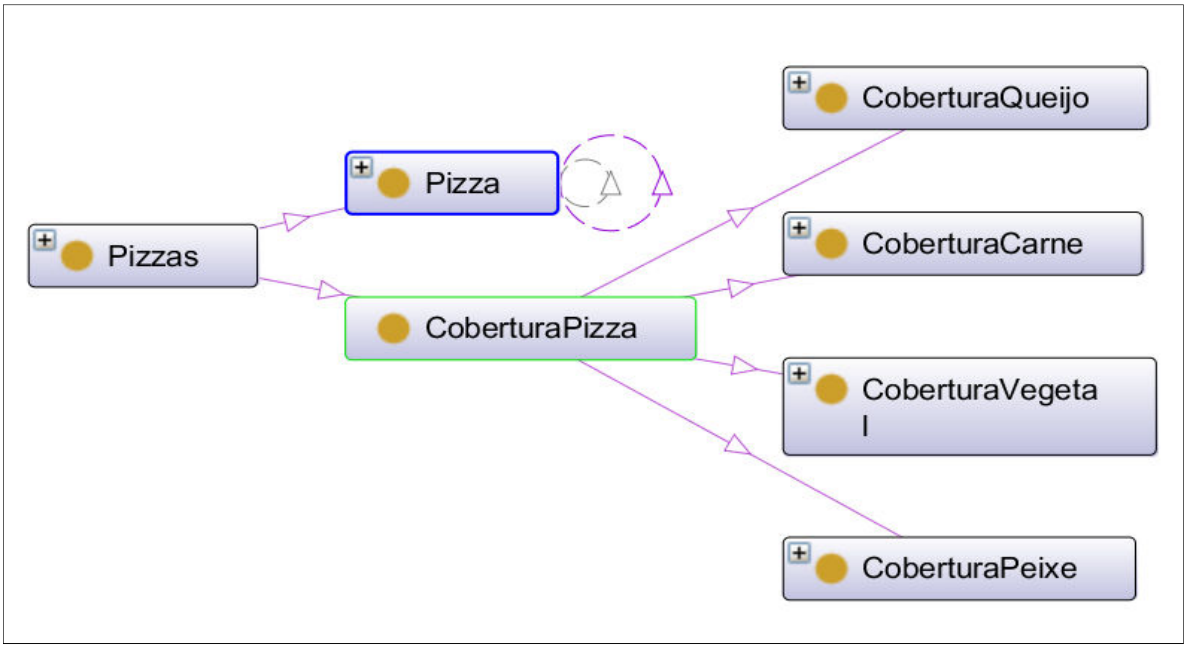


Figura 2 – Classe CoberturaPizza e suas ramificações

Após as divisões das coberturas em 04 categorias, foram inseridos os sabores das coberturas respectivas de cada categoria. A Figura 3 representa as coberturas de queijo que são Parmesão, Quatro Queijos e Mussarela. A Figura 4 representa as coberturas de carne que contém os sabores de Pepperoni, Calabresa e Frango. A Figura 5 representa as coberturas de vegetal sendo composta dos sabores de Pimenta Vermelha, Pimenta, Tomate, Pimenta Verde, Oliva e Alho. E a Figura 6 representa a cobertura de peixe, na qual é contém os sabores de Frutos do Mar, Camarão e Anchova.

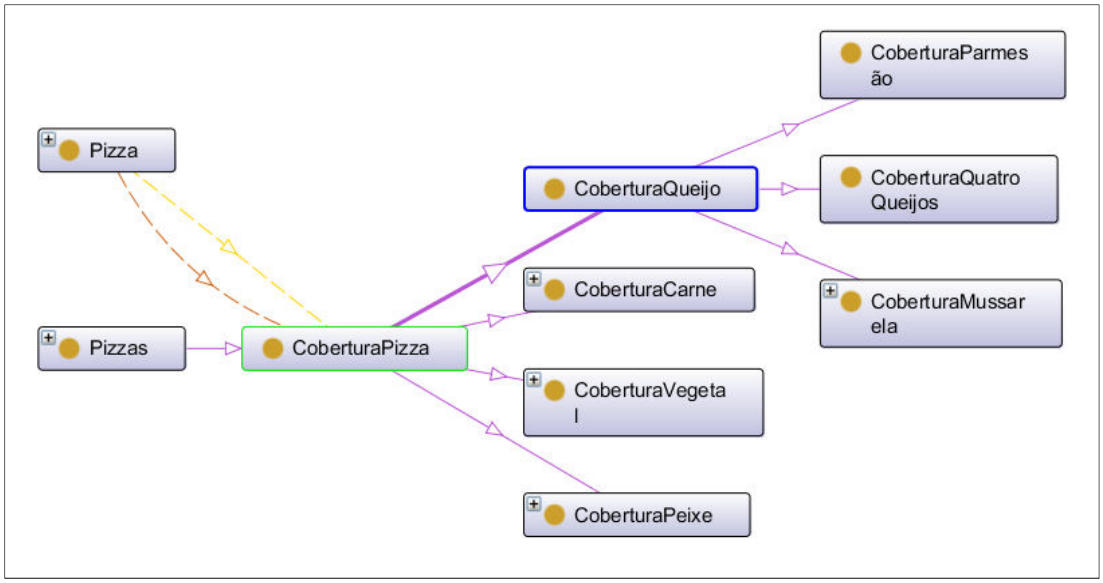


Figura 3 – Ramificação da CoberturaQueijo

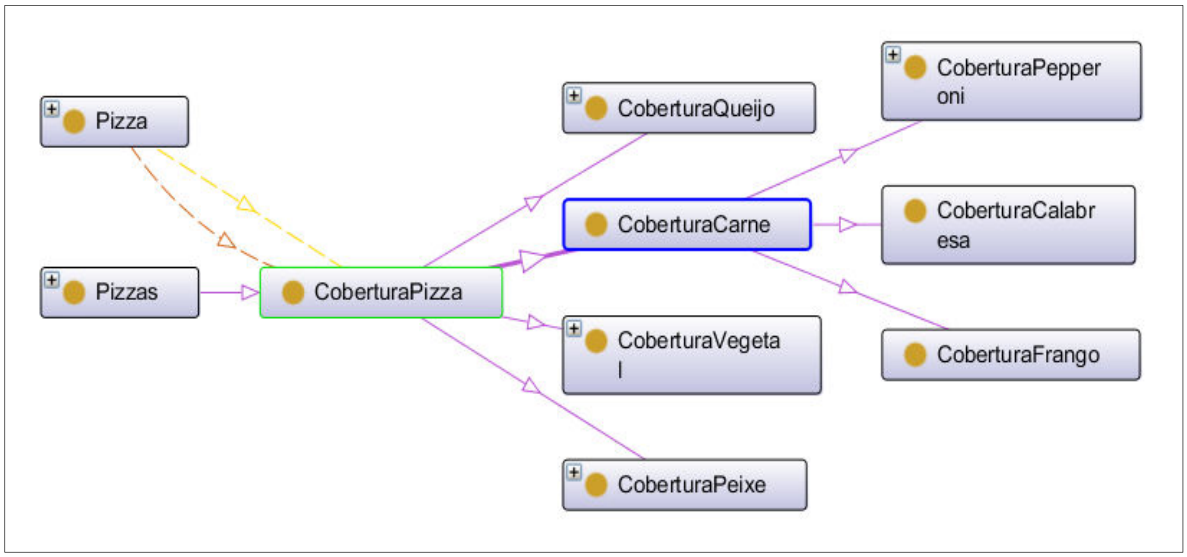


Figura 4 – Ramificação da CoberturaCarne

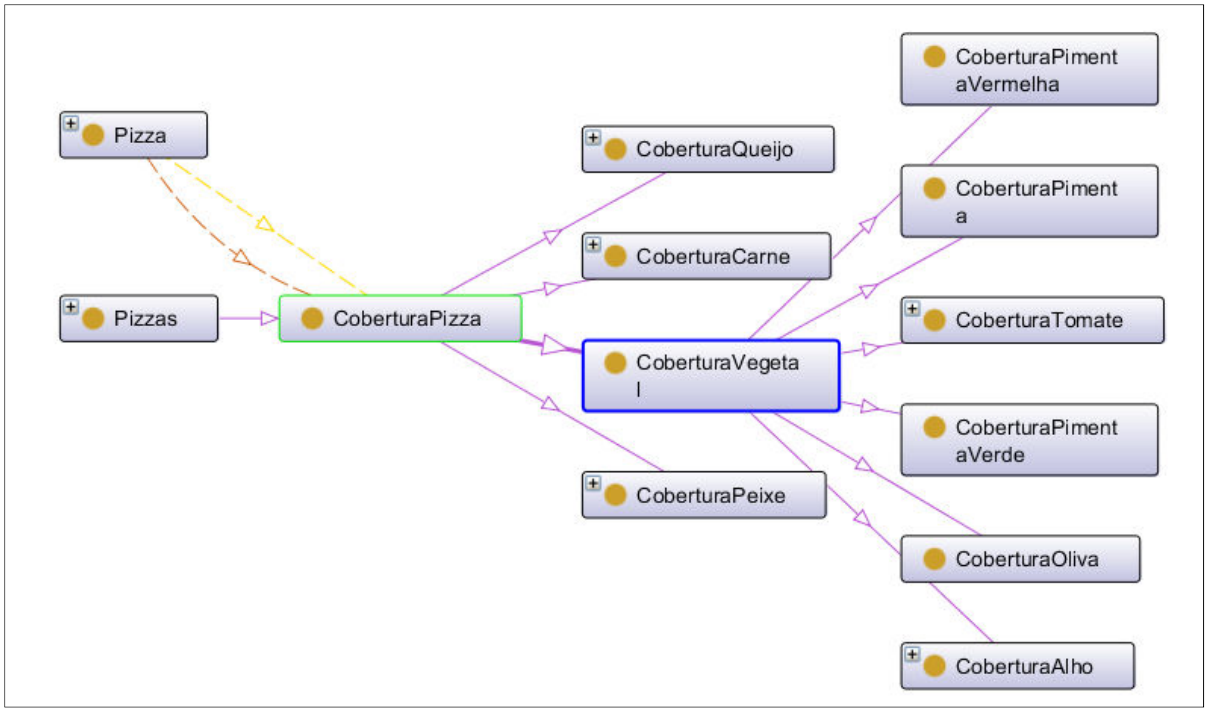


Figura 5 – Ramificação da CoberturaVegetal

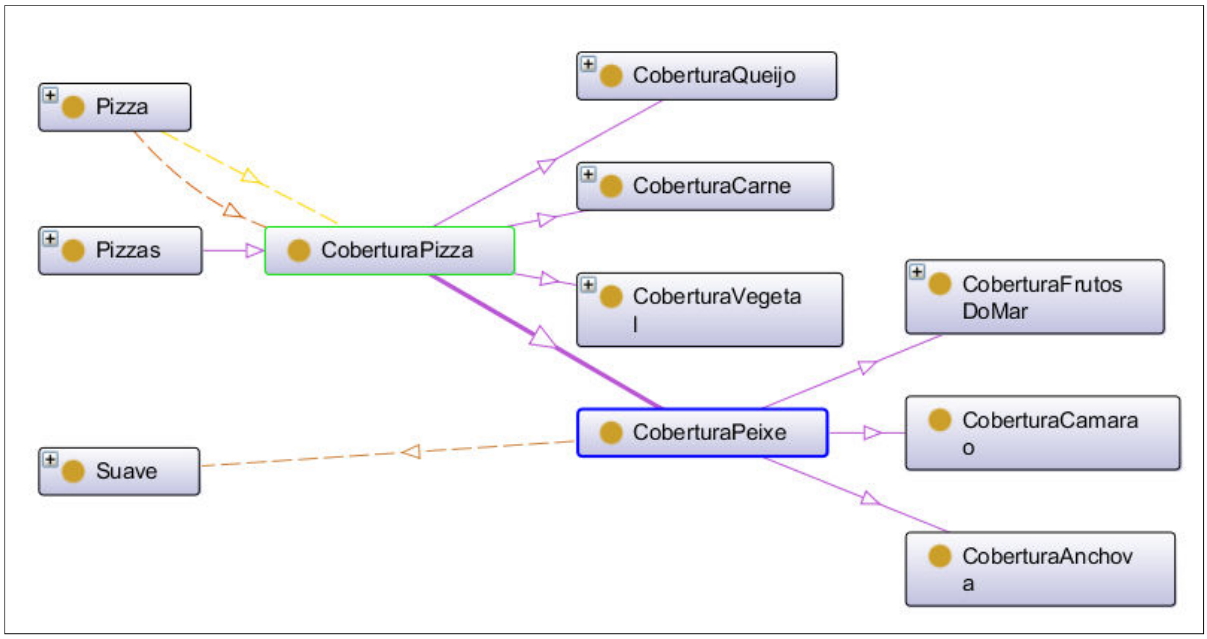


Figura 6 – Ramificação da CoberturaPeixe

Observando a figura 7, vê-se o resultado final da ontologia, na qual a classe Pizzas tem ligação com a classe CoberturaPizza que ramifica em 04 partes diferentes e cada um destes nós ramificam para os nós folhas, neste caso os sabores das coberturas das pizzas.

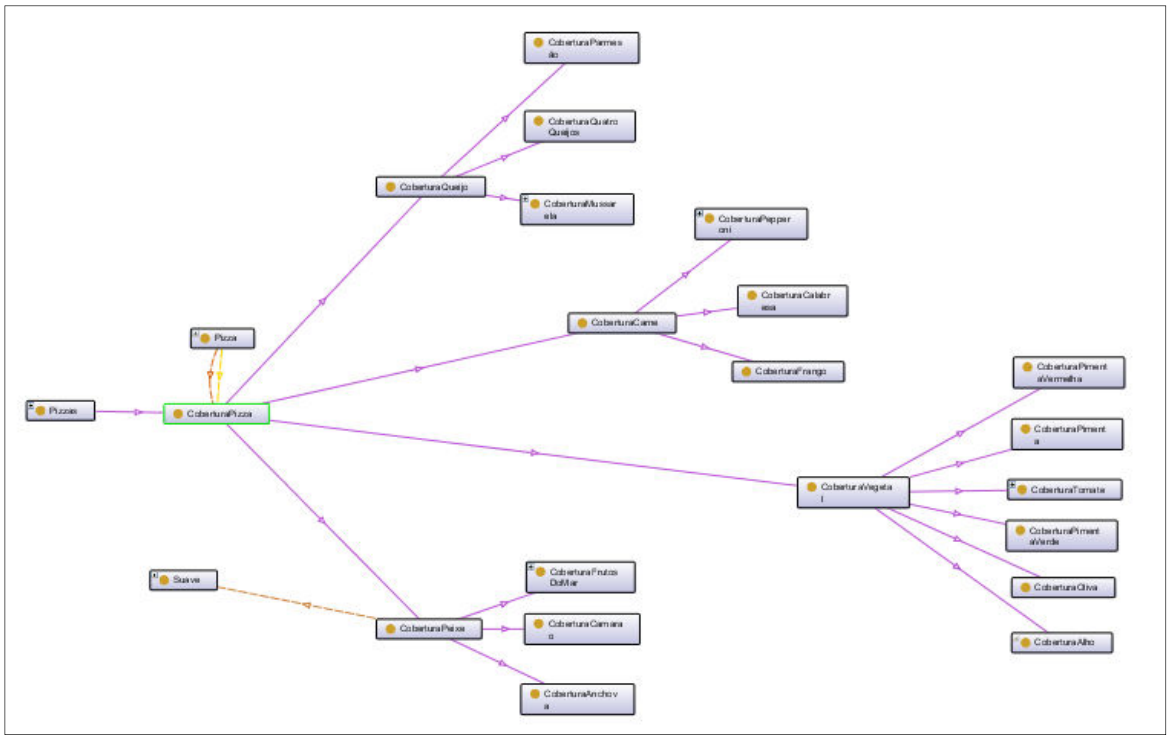


Figura 7 – Diagrama da Ontologia pizza.owl

## 3.3 FUNCIONAMENTO DO CHATTERBOT

A base AIML deve possuir somente respostas para perguntas triviais de uma conversação, tais como formas de cumprimento, identificação do cliente, endereço, entre outras informações para entrega, etc. As informações referentes às pizzas e os sabores das coberturas devem estar descritas na ontologia.

Utilizou-se também da implementação de uma classe Java com o nome OntologiaPizza (Anexo B) para acessar as informações contidas em ontologias OWL, que tem como objetivo ser utilizada como uma camada de abstração entre os diálogos AIML e a ontologia. O funcionamento básico da classe OntologiaPizza (Anexo B) se dá quando um dialogo AIML pode acessá-la, solicitando informações, sem necessariamente saber como essas informações estão implementadas e

organizadas na ontologia OWL, justificando assim a denominação "camada de abstração".

Existe uma *tag* especial chamada "<*system*>" dentro da especificação AIML, que pode ser implementada também no *Program D*, criando-se assim um canal de comunicação entre o *Program D* e a classe OntologiaPizza (Anexo B), possibilitando a integração do sistema com a ontologia. Ambos, *Program D* e a classe OntologiaPizza (Anexo B), se comunicarão através de requisições. As requisições estarão dentro das *tags* "<*system*>" e "</*system*>" e serão repassadas para a OntologiaPizza (Anexo B) e processadas.

Se o *chatterbot* encontrar um padrão compatível dentro do objeto AIML, uma resposta é devolvida ao usuário.

Porém, se dentro de uma resposta houver a *tag* "<*system*>", ocorre automaticamente uma requisição a OntologiaPizza (Anexo B) através de um método onde está encapsulado uma requisição. A OntologiaPizza (Anexo B) ao processar a requisição devolve o resultado que fica no lugar da *tag* "<*system*>" dentro da resposta do AIML que finalmente é enviado ao usuário.

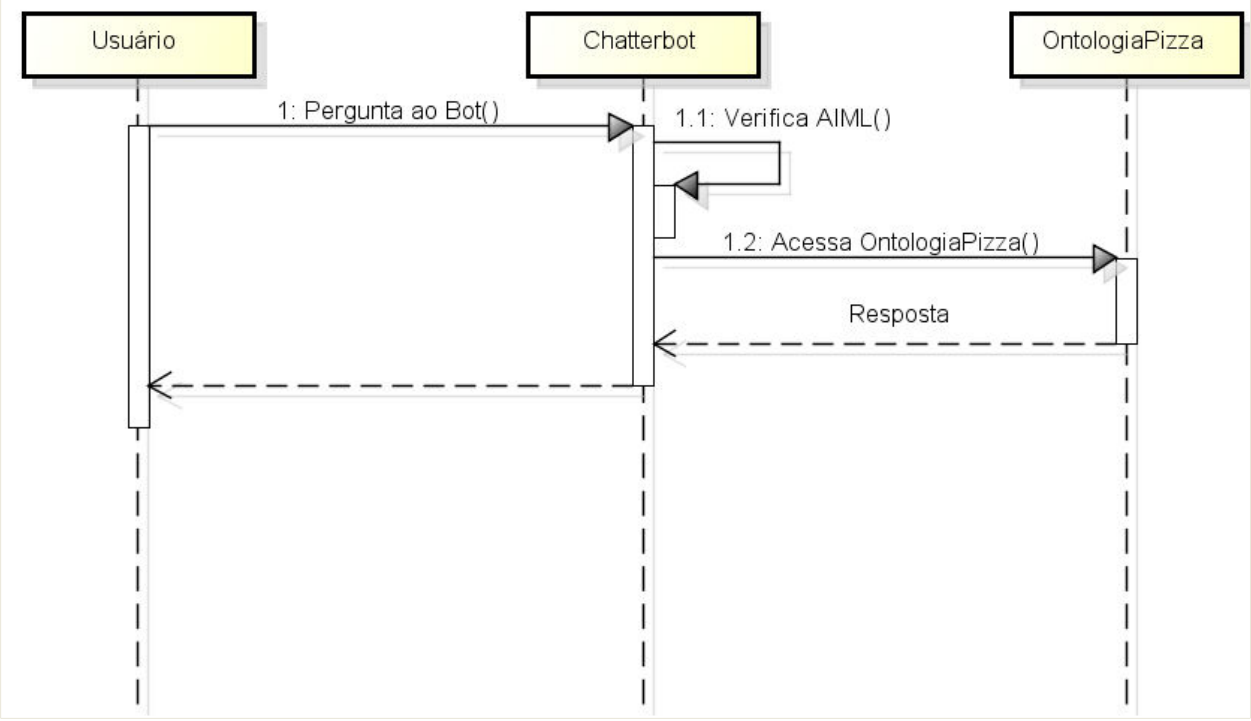


Figura 8 – Modelo do Funcionamento do Program D e da OntologiaPizza

A Figura 9 ilustra o módulo de conversação, exemplificando o modelo de navegação entre usuário, chatterbot e seus subsistemas.

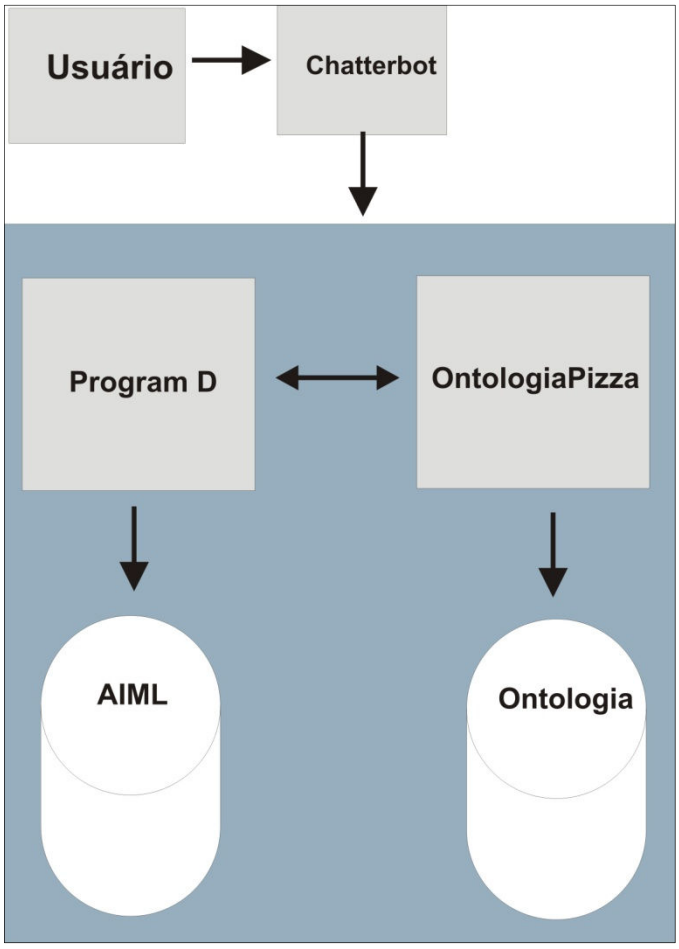


Figura 9 – Módulo de Conversação

## 3.4 TESTES

Com o *chatterbot* em execução e as bases de dados carregadas, foi realizado testes. Foram feito testes de conversação com o chatterbot, testes nos quais constituíram de testar todos os diálogos implementados no *chatterbot* para verificar se quando questionado se o mesmo responderia de acordo com o que foi programado.

Foram também realizados testes para verificar se o mesmo quando informado sobre o saber de uma pizza, acessaria a classe OntologiaPizza (Anexo B) através da *tag* <*system*> e retornaria a pizza correspondente ao sabor solicitado.

Os diálogos utilizados para montar a base AIML funcionaram corretamente retornando todas as respostas registradas. Entretanto, é necessário expandir ainda mais a base, variando as formas que uma mesma pergunta possa ser feita pelo usuário. Uma maneira de atualizar as bases AIML rapidamente é através do *log* gerado pelo *chatterbot* com todas as conversas realizadas entre ele e o usuário.

Entre as dificuldades encontradas no desenvolvimento é a falta de códigos AIML na língua portuguesa, tendo a necessidade de traduzir os diálogos básicos encontrados em inglês para o português. Outra dificuldade encontrada é a pouca de documentação relacionada às *tags* AIML, dificultando a implementação da mesma.

# CONCLUSÃO

Com o desenvolvimento deste projeto pretende facilitar a comunicação entre homem e computador, agregando a linguagem natural nas respostas originárias do sistema. Dessa forma, os sistemas poderão interpretar e retornar informações na linguagem humana, melhorando a interação entre usuário e sistema.

São muitas as vantagens no uso de um *chatterbot*, dentre essas, pode-se citar, estar 24 horas à disposição na Internet para conversar com os usuários, baixo custo de implantação e manutenção, padronização na disseminação da informação, os diálogos podem ser gravados visando uma análise das informações e aprimoramento da base de conhecimento.

Possibilitará futuramente o desenvolvimento de sistemas similares a *chatterbots* capazes de interpretar a linguagem natural utilizada pelo usuário. Permitirá a criação de sistemas que a partir de perguntas feitas pelo usuário, geram bases de dados e retornam as informações requeridas pelo usuário, e até sistemas que conseguem captar comandos por voz, interpretá-los e retornar as informações.

Como quase todos os sistemas, o Bot possui algumas restrições na versão inicial, porém todas as restrições identificadas até o momento não inviabilizam o projeto. Uma das principais restrições é que as bases AIML utilizadas inicialmente não conterão todas as maneiras que uma pessoa possa realizar uma pergunta, já que a língua portuguesa possui várias expressões e gírias para um mesmo significado. Para correção desta é necessário observar o arquivo de *log* das conversas conforme o sistema for sendo utilizado e assim atualizar sua base.

Destaca-se que ao utilizar e testar um *chatterbot* diversos aspectos não convencionais, aos demais sistemas, devem ser considerados tais como a análise do impacto no uso deste tipo de tecnologia e a abrangência do vocabulário.

# REFERENCIAS

CANUTO, Eudes Pimentel. **VICTOR-P: UM CVA CHATTERBOT COM PERSONALIDADE.** 2005. 68 f. Dissertação (Graduação) - Curso de Ciência Da Computação, Departamento de Centro De Informática, Universidade Federal De Pernambuco, Recife, 2005.

FERREIRA, Leonardo. Chatterbots: Trocas Comunicativas em Ambientes Virtuais e Inteligências Artificiais. In: CONGRESSO DE CIÊNCIAS DA COMUNICAÇÃO NA REGIÃO NORDESTE, 10., 2008, Fortaleza. **Tese.** São Luís: Congresso de Ciências da Comunicação Na Região Nordeste, 2008. p. 1 - 15.

LOPES, Ilza Leite. Uso das linguagens controlada e natural em bases de dados: revisão da literatura. **Ci. Inf.**, Brasília, n. , p.41-52, abr. 2002.

MERENIUK, Fabricio Cavassin. **Sistema auto-adaptativo, para acionamento de dispositivos, com linguagem natural.** 2004. 51 f. Monografia (Graduação) - Curso de Engenharia da Computação, Departamento de Núcleo de Ciências Exatas e Tecnológicas - Ncet, Centro Universitário Positivo - Unicenp, Curitiba, 2004.

MOURA, Deise Silva De. **PSICOCHAT: CHATTERBOT APLICADO AO ENSINO DE PSIQUIATRIA.** 2008. 70 f. Dissertação (Graduação) - Curso de Sistemas de Informação, Departamento de Instituto de Ciências Exatas e Tecnológicas, Centro Universitário Feevale, Nova Hamburgo, 2008.

OLIVEIRA NETO, João Mendes de; TONIN, Sávio Duarte; PRIETCH, Soraia Silva. **Processamento de Linguagem Natural e suas Aplicações.** Rondonópolis: Desconhecido, 2004.

PEREIRA, Rui Alexandre da Rocha Gonçalves. **Editor para Web Semântica Integrando Anotações Semânticas, Ontologias e RDF.** 2006. 277 f. Dissertação

(Mestrado) - Curso de Engenharia Informática, Departamento de Informática, Universidade da Beira Interior, Covilhã, 2006.

RAMOS, Josion Das Neves. **FaqBot - Um Sistema de Raciocínio Baseado em Casos com Processamento Linguístico.** 2008. 42 f. Monografia (Graduação) - Curso de Ciência da Computação, Universidade Luterana do Brasil, Gravataí, 2008.

SILVA, José Wendell de Morais. **Recuperação de Informação através de Recursos da Web Semântica: Modelagem e Implementação.** 2007. 41 f. Dissertação (Tecnólogo) - Curso de Tecnologia em Sistemas Para Internet, Centro Federal de Educação Tecnológica da Paraíba - Cefetpb, João Pessoa, 2007.

WANTROBA, Ewerton José et al. **UM EXEMPLO DE USO DO PADRÃO XML NA DEFINIÇÃO DE UMA LINGUAGEM ESPECIALIZADA PARA A INTELIGÊNCIA ARTIFICIAL.** Ponta Grossa: Publ. Uepg Ci. Exatas Terra, Ci. Agr. Eng, 2008.

W3C. **OWL.** Disponível em: <http://www.w3.org/TR/owl-features/>. Acesso em: 28 out. 2011.

# Anexos

### Anexo A - Código da Ontologia

```
<?xml version="1.0"?>

<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/ontologies/2012/7/Pizza.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.semanticweb.org/ontologies/2012/7/Pizza.owl">
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="" IRI="http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Declaration>
        <Class IRI="#Americana"/>
    </Declaration>
    <Declaration>
        <Class IRI="#CoberturaAlho"/>
    </Declaration>
```

```
<Declaration>
    <Class IRI="#CoberturaAnchova"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaCalabresa"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaCamarao"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaCarne"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaFrango"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaFrutosDoMar"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaMussarela"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaOliva"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaParmesão"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPeixe"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPepperoni"/>
```

```
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPimenta"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPimentaVerde"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPimentaVermelha"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaPizza"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaQuatroQueijos"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaQueijo"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaTomate"/>
</Declaration>
<Declaration>
    <Class IRI="#CoberturaVegetal"/>
</Declaration>
<Declaration>
    <Class IRI="#FrutosDoMar"/>
</Declaration>
<Declaration>
    <Class IRI="#Margherita"/>
</Declaration>
<Declaration>
```

```
    <Class IRI="#MassaFina"/>
</Declaration>
<Declaration>
    <Class IRI="#MassaGrossa"/>
</Declaration>
<Declaration>
    <Class IRI="#MassaPizza"/>
</Declaration>
<Declaration>
    <Class IRI="#Medio"/>
</Declaration>
<Declaration>
    <Class IRI="#OutrasPizzas"/>
</Declaration>
<Declaration>
    <Class IRI="#Paises"/>
</Declaration>
<Declaration>
    <Class IRI="#Pizza"/>
</Declaration>
<Declaration>
    <Class IRI="#PizzaCarne"/>
</Declaration>
<Declaration>
    <Class IRI="#PizzaInteressante"/>
</Declaration>
<Declaration>
    <Class IRI="#PizzaItaliana"/>
</Declaration>
<Declaration>
    <Class IRI="#PizzaMussarela"/>
</Declaration>
```

```
<Declaration>
   <Class IRI="#PizzaNaoVegetariana"/>
</Declaration>
<Declaration>
   <Class IRI="#PizzaVegetariana"/>
</Declaration>
<Declaration>
   <Class IRI="#Pizzas"/>
</Declaration>
<Declaration>
   <Class IRI="#Quente"/>
</Declaration>
<Declaration>
   <Class IRI="#Suave"/>
</Declaration>
<Declaration>
   <Class IRI="#Temperos"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#ehCoberturaDe"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#ehIngredienteDe"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#ehMassaDe"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#temCobertura"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#temGostoParecido"/>
```

```
</Declaration>
<Declaration>
    <ObjectProperty IRI="#temIngrediente"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#temMassa"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#temPaisDeOrigem"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#temTempero"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Alemanha"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#America"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Franca"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Inglaterra"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Italia"/>
</Declaration>
<EquivalentClasses>
    <Class IRI="#PizzaCarne"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
```

```
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#temCobertura"/>
            <Class IRI="#CoberturaCarne"/>
        </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PizzaInteressante"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
        <ObjectMinCardinality cardinality="3">
            <ObjectProperty IRI="#temCobertura"/>
        </ObjectMinCardinality>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PizzaItaliana"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
        <ObjectHasValue>
            <ObjectProperty IRI="#temPaisDeOrigem"/>
            <NamedIndividual IRI="#Italia"/>
        </ObjectHasValue>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PizzaMussarela"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#temCobertura"/>
            <Class IRI="#CoberturaQueijo"/>
```

```
        </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PizzaNaoVegetariana"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
        <ObjectComplementOf>
            <Class IRI="#PizzaVegetariana"/>
        </ObjectComplementOf>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#PizzaVegetariana"/>
    <ObjectIntersectionOf>
        <Class IRI="#Pizza"/>
        <ObjectComplementOf>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#temCobertura"/>
                <Class IRI="#CoberturaCarne"/>
            </ObjectSomeValuesFrom>
        </ObjectComplementOf>
        <ObjectComplementOf>
            <ObjectSomeValuesFrom>
                <ObjectProperty IRI="#temCobertura"/>
                <Class IRI="#CoberturaPeixe"/>
            </ObjectSomeValuesFrom>
        </ObjectComplementOf>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#Temperos"/>
```

```
    <ObjectUnionOf>
        <Class IRI="#Medio"/>
        <Class IRI="#Quente"/>
        <Class IRI="#Suave"/>
    </ObjectUnionOf>
</EquivalentClasses>
<SubClassOf>
    <Class IRI="#Americana"/>
    <Class IRI="#OutrasPizzas"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Americana"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaMussarela"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Americana"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaPepperoni"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Americana"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaTomate"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
```

```
    <Class IRI="#Americana"/>
    <ObjectAllValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <ObjectUnionOf>
            <Class IRI="#CoberturaMussarela"/>
            <Class IRI="#CoberturaPepperoni"/>
            <Class IRI="#CoberturaTomate"/>
        </ObjectUnionOf>
    </ObjectAllValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Americana"/>
    <ObjectHasValue>
        <ObjectProperty IRI="#temPaisDeOrigem"/>
        <NamedIndividual IRI="#America"/>
    </ObjectHasValue>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaAlho"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaAlho"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temTempero"/>
        <Class IRI="#Medio"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaAnchova"/>
    <Class IRI="#CoberturaPeixe"/>
</SubClassOf>
```

```
<SubClassOf>
    <Class IRI="#CoberturaCalabresa"/>
    <Class IRI="#CoberturaCarne"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaCamarao"/>
    <Class IRI="#CoberturaPeixe"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaCarne"/>
    <Class IRI="#CoberturaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaFrango"/>
    <Class IRI="#CoberturaCarne"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaFrutosDoMar"/>
    <Class IRI="#CoberturaPeixe"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaMussarela"/>
    <Class IRI="#CoberturaQueijo"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaOliva"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaParmesão"/>
    <Class IRI="#CoberturaQueijo"/>
</SubClassOf>
```

```
<SubClassOf>
    <Class IRI="#CoberturaPeixe"/>
    <Class IRI="#CoberturaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPeixe"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temTempero"/>
        <Class IRI="#Suave"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPepperoni"/>
    <Class IRI="#CoberturaCarne"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPimenta"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPimentaVerde"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPimentaVermelha"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaPizza"/>
    <Class IRI="#Pizzas"/>
</SubClassOf>
<SubClassOf>
```

```
    <Class IRI="#CoberturaQuatroQueijos"/>
    <Class IRI="#CoberturaQueijo"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaQueijo"/>
    <Class IRI="#CoberturaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaTomate"/>
    <Class IRI="#CoberturaVegetal"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#CoberturaVegetal"/>
    <Class IRI="#CoberturaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#FrutosDoMar"/>
    <Class IRI="#OutrasPizzas"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#FrutosDoMar"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaAlho"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#FrutosDoMar"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaFrutosDoMar"/>
    </ObjectSomeValuesFrom>
```

```
</SubClassOf>
<SubClassOf>
   <Class IRI="#FrutosDoMar"/>
   <ObjectSomeValuesFrom>
      <ObjectProperty IRI="#temCobertura"/>
      <Class IRI="#CoberturaTomate"/>
   </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
   <Class IRI="#FrutosDoMar"/>
   <ObjectAllValuesFrom>
      <ObjectProperty IRI="#temCobertura"/>
      <ObjectUnionOf>
         <Class IRI="#CoberturaAlho"/>
         <Class IRI="#CoberturaFrutosDoMar"/>
         <Class IRI="#CoberturaTomate"/>
      </ObjectUnionOf>
   </ObjectAllValuesFrom>
</SubClassOf>
<SubClassOf>
   <Class IRI="#Margherita"/>
   <Class IRI="#OutrasPizzas"/>
</SubClassOf>
<SubClassOf>
   <Class IRI="#Margherita"/>
   <ObjectSomeValuesFrom>
      <ObjectProperty IRI="#temCobertura"/>
      <Class IRI="#CoberturaMussarela"/>
   </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
   <Class IRI="#Margherita"/>
```

```
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <Class IRI="#CoberturaTomate"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Margherita"/>
    <ObjectAllValuesFrom>
        <ObjectProperty IRI="#temCobertura"/>
        <ObjectUnionOf>
            <Class IRI="#CoberturaMussarela"/>
            <Class IRI="#CoberturaTomate"/>
        </ObjectUnionOf>
    </ObjectAllValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#MassaFina"/>
    <Class IRI="#MassaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#MassaGrossa"/>
    <Class IRI="#MassaPizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#MassaPizza"/>
    <Class IRI="#Pizzas"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Medio"/>
    <Class IRI="#Temperos"/>
</SubClassOf>
<SubClassOf>
```

```
    <Class IRI="#OutrasPizzas"/>
    <Class IRI="#Pizza"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Pizza"/>
    <Class IRI="#Pizzas"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Pizza"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#temMassa"/>
        <Class IRI="#MassaPizza"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#PizzaItaliana"/>
    <ObjectAllValuesFrom>
        <ObjectProperty IRI="#temMassa"/>
        <Class IRI="#MassaFina"/>
    </ObjectAllValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Quente"/>
    <Class IRI="#Temperos"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Suave"/>
    <Class IRI="#Temperos"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Temperos"/>
    <Class abbreviatedIRI="owl:Thing"/>
```

```
</SubClassOf>
<DisjointClasses>
    <Class IRI="#Americana"/>
    <Class IRI="#FrutosDoMar"/>
    <Class IRI="#Margherita"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Americana"/>
    <Class IRI="#Margherita"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaAlho"/>
    <Class IRI="#CoberturaOliva"/>
    <Class IRI="#CoberturaPimenta"/>
    <Class IRI="#CoberturaPimentaVerde"/>
    <Class IRI="#CoberturaPimentaVermelha"/>
    <Class IRI="#CoberturaTomate"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaAnchova"/>
    <Class IRI="#CoberturaCamarao"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaAnchova"/>
    <Class IRI="#CoberturaFrutosDoMar"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaCalabresa"/>
    <Class IRI="#CoberturaFrango"/>
    <Class IRI="#CoberturaPepperoni"/>
</DisjointClasses>
<DisjointClasses>
```

```
    <Class IRI="#CoberturaCarne"/>
    <Class IRI="#CoberturaPeixe"/>
    <Class IRI="#CoberturaQueijo"/>
    <Class IRI="#CoberturaVegetal"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaMussarela"/>
    <Class IRI="#CoberturaParmesão"/>
    <Class IRI="#CoberturaQuatroQueijos"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#CoberturaPizza"/>
    <Class IRI="#MassaPizza"/>
    <Class IRI="#Pizza"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#MassaFina"/>
    <Class IRI="#MassaGrossa"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Medio"/>
    <Class IRI="#Quente"/>
    <Class IRI="#Suave"/>
</DisjointClasses>
<ClassAssertion>
    <Class IRI="#Paises"/>
    <NamedIndividual IRI="#Alemanha"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI="owl:Thing"/>
    <NamedIndividual IRI="#Alemanha"/>
</ClassAssertion>
```

```
<ClassAssertion>
    <Class IRI="#Paises"/>
    <NamedIndividual IRI="#America"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI="owl:Thing"/>
    <NamedIndividual IRI="#America"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Paises"/>
    <NamedIndividual IRI="#Franca"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI="owl:Thing"/>
    <NamedIndividual IRI="#Franca"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Paises"/>
    <NamedIndividual IRI="#Inglaterra"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI="owl:Thing"/>
    <NamedIndividual IRI="#Inglaterra"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Paises"/>
    <NamedIndividual IRI="#Italia"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI="owl:Thing"/>
    <NamedIndividual IRI="#Italia"/>
</ClassAssertion>
```

```
<DifferentIndividuals>
    <NamedIndividual IRI="#Alemanha"/>
    <NamedIndividual IRI="#America"/>
    <NamedIndividual IRI="#Franca"/>
    <NamedIndividual IRI="#Inglaterra"/>
    <NamedIndividual IRI="#Italia"/>
</DifferentIndividuals>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#ehCoberturaDe"/>
    <ObjectProperty IRI="#ehIngredienteDe"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#ehMassaDe"/>
    <ObjectProperty IRI="#ehIngredienteDe"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#temCobertura"/>
    <ObjectProperty IRI="#temIngrediente"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#temMassa"/>
    <ObjectProperty IRI="#temIngrediente"/>
</SubObjectPropertyOf>
<InverseObjectProperties>
    <ObjectProperty IRI="#ehCoberturaDe"/>
    <ObjectProperty IRI="#temCobertura"/>
</InverseObjectProperties>
<InverseObjectProperties>
    <ObjectProperty IRI="#ehIngredienteDe"/>
    <ObjectProperty IRI="#temIngrediente"/>
</InverseObjectProperties>
<InverseObjectProperties>
```

```
    <ObjectProperty IRI="#ehMassaDe"/>
    <ObjectProperty IRI="#temMassa"/>
</InverseObjectProperties>
<FunctionalObjectProperty>
    <ObjectProperty IRI="#temMassa"/>
</FunctionalObjectProperty>
<InverseFunctionalObjectProperty>
    <ObjectProperty IRI="#ehMassaDe"/>
</InverseFunctionalObjectProperty>
<SymmetricObjectProperty>
    <ObjectProperty IRI="#temGostoParecido"/>
</SymmetricObjectProperty>
<TransitiveObjectProperty>
    <ObjectProperty IRI="#ehIngredienteDe"/>
</TransitiveObjectProperty>
<TransitiveObjectProperty>
    <ObjectProperty IRI="#temIngrediente"/>
</TransitiveObjectProperty>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#ehCoberturaDe"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#ehIngredienteDe"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#ehMassaDe"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#temCobertura"/>
```

```
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#temIngrediente"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#temMassa"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI="#ehCoberturaDe"/>
    <Class IRI="#CoberturaPizza"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#ehIngredienteDe"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#ehMassaDe"/>
    <Class IRI="#MassaPizza"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#temCobertura"/>
    <Class IRI="#CoberturaPizza"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#temIngrediente"/>
    <Class IRI="#Pizza"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#temMassa"/>
```

```
        <Class IRI="#MassaPizza"/>
    </ObjectPropertyRange>
</Ontology>

<!-- Generated by the OWL API (version 3.2.3.1824) http://owlapi.sourceforge.net -->
```

**Anexo B – Código da Classe da OntologiaPizza**

```
package org.aitools.programd;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import org.semanticweb.HermiT.Reasoner.ReasonerFactory;

import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.io.OWLOntologyCreationIOException;
import org.semanticweb.owlapi.io.OWLParser;
import org.semanticweb.owlapi.io.OWLParserException;
import org.semanticweb.owlapi.io.UnparsableOntologyException;
import org.semanticweb.owlapi.model.*;
import org.semanticweb.owlapi.reasoner.ConsoleProgressMonitor;
import org.semanticweb.owlapi.reasoner.Node;
import org.semanticweb.owlapi.reasoner.NodeSet;
import org.semanticweb.owlapi.reasoner.OWLReasoner;
import org.semanticweb.owlapi.reasoner.OWLReasonerConfiguration;
import org.semanticweb.owlapi.reasoner.OWLReasonerFactory;
import org.semanticweb.owlapi.reasoner.SimpleConfiguration;
import org.semanticweb.owlapi.util.DefaultPrefixManager;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
/**
 *
 * @author Thiago
 */
public class OntologiaPizza {

    public String OntologiaPizza(String pizzaSabor){

        String resposta = null;

        try {

            String ontoPizza = "C:/Users/Thiago/Documents/Faculdade/Pizza/Pizza.owl";
            String saborPizza = pizzaSabor;

            OWLOntologyManager gerente =
OWLManager.createOWLOntologyManager();

            File file = new File(ontoPizza);

            OWLOntology local = gerente.loadOntologyFromOntologyDocument(file);

            IRI documentIRI = gerente.getOntologyDocumentIRI(local);
            System.out.println("De: " + documentIRI);

            OWLReasonerFactory reasonerFactory = new ReasonerFactory();

            //Crie um monitor de progresso console. Isto irá imprimir o progresso
raciocinador fora para o console.
            ConsoleProgressMonitor progressMonitor = new ConsoleProgressMonitor();
```

```
        //Especificar o monitor de progresso através de uma configuração.
        OWLReasonerConfiguration config = new
SimpleConfiguration(progressMonitor);

        //Criar um pensador que vai raciocinar sobre a nossa ontologia e seu
fechamento importações. Passe na configuração.
        OWLReasoner reasoner = reasonerFactory.createReasoner(local, config);

        //Pergute ao reasoner
        reasoner.precomputeInferences();

        //Podemos determinar se a ontologia é consistente
        boolean consistent = reasoner.isConsistent();
        System.out.println("Consistencia: " + consistent);
        System.out.println("\n");

        /*Node<OWLClass> bottomNode = reasoner.getUnsatisfiableClasses();

        Set<OWLClass> unsatisfiable = bottomNode.getEntitiesMinusBottom();
        if (!unsatisfiable.isEmpty()) {
            System.out.println("As seguites classes são insatisfatórias: ");
            for (OWLClass cls : unsatisfiable) {
                System.out.println(" " + cls);
            }
        } else {
            System.out.println("Todas as classes estão insatisfatórias");
        }
        System.out.println("\n");*/
        if ("carne".equals(saborPizza)){
         //Query reasoner
        OWLDataFactory fac = gerente.getOWLDataFactory();
```

```
        //Obter referencias
        OWLClass query = fac.getOWLClass(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#CoberturaCarne"));

        //Agora utilize o reasoner para obter a subClasse
        //Podemos pedir as subclasses diretas, para isso utiliza-se o true
        NodeSet<OWLClass> subClases = reasoner.getSubClasses(query, true);

        /*O raciocinador retorna uma NodeSet, o que representa um conjunto de nodos.
        Cada nó no conjunto representa uma subclasse. Um nó de classes contém classes,
        onde cada classe no nó é equivalente. Por exemplo, se pedimos para as subclasses de
        alguma classe A e voltou um NodeSet contendo dois nós {B, C} e {D}, então A teria
        duas subclasses apropriadas. Um destes subclasses seria equivalente à classe D, e o
        outro seria a classe que é equivalente à classe B e classe C*/

        Set<OWLClass> clses = subClases.getFlattened();
        //System.out.println("Subclasses: ");
        for (OWLClass cls : clses) {
            System.out.println(" " + pm.getShortForm(cls));
            resposta = pm.getShortForm(cls);
        }
        //resposta = clses.toString();
        System.out.println("\n");

        /*String resposta = subClases.getFlattened().toString();
        System.out.println(resposta);*/
```

```
        }
        else if ("peixe".equals(saborPizza)){
            //Query reasoner
        OWLDataFactory fac = gerente.getOWLDataFactory();

        //Obter referencias
        OWLClass query =
fac.getOWLClass(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#CoberturaPeixe"));

        //Agora utilize o reasoner para obter a subClasse
        //Podemos pedir as subclasses diretas, para isso utiliza-se o true
        NodeSet<OWLClass> subClases = reasoner.getSubClasses(query, true);

         /*O raciocinador retorna uma NodeSet, o que representa um conjunto de nodos.
        Cada nó no conjunto representa uma subclasse. Um nó de classes contém classes,
        onde cada classe no nó é equivalente. Por exemplo, se pedimos para as subclasses de
        alguma classe A e voltou um NodeSet contendo dois nós {B, C} e {D}, então A teria
        duas subclasses apropriadas. Um destes subclasses seria equivalente à classe D, e o
        outro seria a classe que é equivalente à classe B e classe C*/

        Set<OWLClass> clses = subClases.getFlattened();
        //System.out.println("Subclasses: ");
        for (OWLClass cls : clses) {
            System.out.println(" " + pm.getShortForm(cls));
            resposta = pm.getShortForm(cls);
        }
```

```
//resposta = clses.toString();

System.out.println("\n");
}
else if ("queijo".equals(saborPizza)){
   //Query reasoner
OWLDataFactory fac = gerente.getOWLDataFactory();

//Obter referencias
OWLClass query = fac.getOWLClass(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#CoberturaQueijo"));

//Agora utilize o reasoner para obter a subClasse
//Podemos pedir as subclasses diretas, para isso utiliza-se o true
NodeSet<OWLClass> subClases = reasoner.getSubClasses(query, true);

 /*O raciocinador retorna uma NodeSet, o que representa um conjunto de nodos.
Cada nó no conjunto representa uma subclasse. Um nó de classes contém classes,
onde cada classe no nó é equivalente. Por exemplo, se pedimos para as subclasses de
alguma classe A e voltou um NodeSet contendo dois nós {B, C} e {D}, então A teria
duas subclasses apropriadas. Um destes subclasses seria equivalente à classe D, e o
outro seria a classe que é equivalente à classe B e classe C*/

Set<OWLClass> clses = subClases.getFlattened();
//System.out.println("Subclasses: ");
for (OWLClass cls : clses) {
```

```
            System.out.println(" " + pm.getShortForm(cls));
            resposta = pm.getShortForm(cls);
        }
        //resposta = clses.toString();

        System.out.println("\n");
        }
        else if ("vegetal".equals(saborPizza)){
            //Query reasoner
        OWLDataFactory fac = gerente.getOWLDataFactory();

        //Obter referencias
        OWLClass query =
fac.getOWLClass(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.o
wl#CoberturaVegetal"));

        //Agora utilize o reasoner para obter a subClasse
        //Podemos pedir as subclasses diretas, para isso utiliza-se o true
        NodeSet<OWLClass> subClases = reasoner.getSubClasses(query, true);

         /*O raciocinador retorna uma NodeSet, o que representa um conjunto de
nodos.
        Cada nó no conjunto representa uma subclasse. Um nó de classes contém
classes,
        onde cada classe no nó é equivalente. Por exemplo, se pedimos para as
subclasses de
        alguma classe A e voltou um NodeSet contendo dois nós {B, C} e {D}, então
A teria
        duas subclasses apropriadas. Um destes subclasses seria equivalente à
classe D, e o
        outro seria a classe que é equivalente à classe B e classe C*/
```

```
        Set<OWLClass> clses = subClases.getFlattened();
        //System.out.println("Subclasses: ");
        for (OWLClass cls : clses) {
           System.out.println(" " + pm.getShortForm(cls));
           resposta = pm.getShortForm(cls);
        }
        //resposta = clses.toString();

        System.out.println("\n");
        }

        /*OWLClass country =
fac.getOWLClass(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.o
wl#Paises"));

        //Perguta sobre as instâncias
        NodeSet<OWLNamedIndividual> individualsNodeSet =
reasoner.getInstances(country, true);

        //O raciocinador retorna NodeSet que contem individuos
        Set<OWLNamedIndividual> individuals = individualsNodeSet.getFlattened();
        System.out.println("Instâncias: ");
        for (OWLNamedIndividual ind : individuals) {
           System.out.println(" " + ind);
        }
        System.out.println("\n");

        //Obter valores de propriedades de um determiando individuo
        OWLNamedIndividual refIndividual =
fac.getOWLNamedIndividual(IRI.create("http://www.semanticweb.org/ontologies/201
2/7/Pizza.owl#PizzaNaoVegetariana"));
```

```
        OWLObjectProperty prop = fac.getOWLObjectProperty(IRI.create("http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#ehIngredienteDe"));

        //Pergunta ao raciocinador sobre a propriedade
        NodeSet<OWLNamedIndividual> propValorNodeSet = reasoner.getObjectPropertyValues(refIndividual, prop);
        Set<OWLNamedIndividual> valor = propValorNodeSet.getFlattened();
        System.out.println("O(s) valor(es) da(s) propriedade(s) é(são): ");
        for (OWLNamedIndividual ind : valor) {
            System.out.println(" " + ind);
        }

        //Imprimir hierarquia da classe
        Node<OWLClass> topNode = reasoner.getTopClassNode();
        print(topNode, reasoner, 0);*/

    } catch (OWLOntologyCreationIOException e){
        IOException ioException = e.getCause();
        if (ioException instanceof FileNotFoundException){
            System.out.println("Não foi possível carregar a ontologia. Arquivo não encontrado: " + ioException.getMessage());
        } else if (ioException instanceof UnknownHostException){
            System.out.println("Não foi possível carregar a ontologia. Host desconhecido: " + ioException.getMessage());
        } else {
            System.out.println("Não foi possível carregar a ontologia: " + ioException.getClass().getSimpleName() + " " + ioException.getMessage());
        }
    } catch (UnparsableOntologyException e){
        System.out.println("Não foi possível parse da ontologia: " + e.getMessage());
```

```
            Map<OWLParser, OWLParserException> exceptions = e.getExceptions();

            for (OWLParser parser : exceptions.keySet()){
                System.out.println("Tentei analisar a ontologia com o " +
parser.getClass().getSimpleName() + "parser");
                System.out.println("Falhou porque: " +
exceptions.get(parser).getMessage());
            }
        } catch (UnloadableImportException e){
            System.out.println("Não foi possível carregar importe: " +
e.getImportsDeclaration());

            OWLOntologyCreationException cause = e.getOntologyCreationException();
            System.out.println("Razão: " + cause.getMessage());
        } catch (OWLOntologyCreationException e){
            System.out.println("Não foi possível carregar a ontologia: " +
e.getMessage());
        }
        return resposta;
    }

     private static void print(Node<OWLClass> parent, OWLReasoner reasoner, int
depth) {
        //Não imprimir nó inferior
        if (parent.isBottomNode()) {
            return;
        }
        //Imprimir um travessão para indicar relações pai-filho
        printIndent(depth);
        //Imprimir o nó
        printNode(parent);
```

```
        for (Node<OWLClass> child :
reasoner.getSubClasses(parent.getRepresentativeElement(), true)) {
            print(child, reasoner, depth + 1);
        }
    }

    private static void printIndent(int depth) {
        for (int i = 0; i < depth; i++) {
            System.out.print(" ");
        }
    }
    private static DefaultPrefixManager pm = new
DefaultPrefixManager("http://www.semanticweb.org/ontologies/2012/7/Pizza.owl#");

    private static void printNode(Node<OWLClass> node) {
        //Imprima um nó como uma lista de nomes de classe
        System.out.println("{");
        for (Iterator<OWLClass> it = node.getEntities().iterator(); it.hasNext();) {
            OWLClass cls = it.next();
            //Usuário gerente de prefixo para fornecer um nome um pouco mais
agradável
            System.out.print(pm.getShortForm(cls));
            if (it.hasNext()) {
                System.out.print(" ");
            }
        }
        System.out.println("}");
    }
}
```