Universidade Federal do Piauí
Campus Senador Helvidio Nunes de Barros
Curso Bacharelado em Sistemas de Informação

Abimael Honório Correia Júnior

SYSTEMBOT – Um *Chatterbot* que fornece informações sobre o curso de Sistemas de Informação - UFPI (CSHNB)

Picos
2014

Abimael Honório Correia Júnior

SYSTEMBOT – Um *Chatterbot* que fornece informações sobre o curso de Sistemas de Informação - UFPI (CSHNB)

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação do Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte para obtenção do Grau de Bacharel em Sistemas de Informação.

Área de Concentração: Inteligência Artificial
Orientador: Dennis Sávio Martins da Silva

Picos
2014

Eu, **Abimael Honório Correia Júnior,** abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI, 14 de agosto de 2014.

Abimael Honório Correia Júnior
Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

C824s Correia Junior, Abimael Honório.
Systembot: Um *chatterbot* que fornece informações sobre o curso de Sistemas de Informação – UFPI (CSHNB) / Abimael Honório Correia Júnior. – 2014.
CD-ROM: il.; 4 ¾ pol. (90 p.)

Monografia (Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2014.
Orientador(A): Prof. Esp. Dennis Sávio Martins da Silva

1. Inteligência Artificial. 2. Agentes de Conversação. 3. Interação Homem – Computador. I. Título.

CDD 006.3

Abimael Honório Correia Júnior

SYSTEMBOT – Um *Chatterbot* que fornece informações sobre o curso de Sistemas de Informação - UFPI (Campus Picos)

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação do Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte para obtenção do Grau de Bacharel em Sistemas de Informação.

Área de Concentração: Inteligência Artificial
Orientador: Dennis Sávio Martins da Silva

Data de Aprovação: 05/08/2014

Prof. Dennis Sávio Martins da Silva Dennis Sávio M. da Silva UFPI - CSHNB
Msc. Alcilene Dalília de Sousa Alcilene Dalília de Sousa UFPI - CSHNB
Msc. Romuere Rodrigues Veloso e Silva UFPI - CSHNB

Picos
2014

Dedico este trabalho primeiramente a Deus, por me dar força e determinação para concluir este trabalho.
Diante de tantos desafios, e acontecimentos que nos fazem pensar em desistir do que mais queremos, percebi que tudo na vida tem um propósito, e cabe a nós descobrirmos qual esse propósito, procurando sempre ter esperança e nunca desistir do que mais queremos.
Dedico também a minha Mãe, Maria do Socorro Vieira da Costa e meu Pai Abimael Honório Correia, aos meus irmãos Amaryel, Amaurya e Kilve, e a Bruna Maia, que se colocaram ao meu lado nos momentos mais difíceis e me deram forças e muito apoio para concluir essa etapa importante de minha vida. Vocês foram essenciais!

AGRADECIMENTOS

Agradeço primordialmente a Deus pelo dom precioso da vida. Esta vida que generosamente me deste e pela qual me sinto responsável. Aos meus pais Abimael e Socorro, que me deram e me ensinaram a vivê-la com dignidade. Euda e Jurema (tias por opção e amor). Não tenho palavras para agradecer todo carinho, compreensão, apoio e conselhos que vocês mês deram durante toda essa jornada.

Aos meus amigos e demais familiares, Zulma, Irany, Alcino Sá, Herlane, Fonseca, Regina, Chico Pio IX, Ziel, Dr. Nilberto, Valéria Liboreiro, Solenita, Silvelene, Lauanny Cardoso, Joshulmar Neiva, Gorete, Maria Floraci, Celso, Fonseca, Rayla Rocha, Oselias Lima e ao meu orientador Dennis Sávio, que sempre acreditaram no meu sucesso e todos aqueles que me acompanharam e que sonharam viver este dia e que agora estão felizes pela minha vitória. Agradeço também, Frei José (in memoriam), Pedro Benvindo (in memoriam), Gilberto (in memoriam), aos meus avós Herozino e Maria (in memoriam), José Santório (in memoriam). Esta conquista é também dedicada a vocês, que embora estrelas do céu, estavam ao meu lado, lutando e iluminando-me. Esta vitória é nossa!

Agradeço aos meus professores; pois agradecer é admitir que sempre precisamos um do outro; é reconhecer que o homem jamais poderá dizer-se autossuficiente. Ninguém cresce sozinho; sempre é preciso olhar de apoio, uma palavra de incentivo, um gesto de compreensão. Professores, a gratidão que hoje lhes reservo é inconfundível, pois não há presente mais edificante ao homem do que o conhecimento. Para mim, vocês foram mais que simples professores; foram amigos. Tornaram-me não só um bom profissional, mais, sobretudo um ser humano melhor. Muito mais que os SISTEMAS DE INFORMAÇÃO, me foi ensinado a arte de cuidar, a humildade de quem se presta a ajudar generosamente através da tecnologia. Por tais ensinamentos, a minha profunda gratidão e respeito. Cada um dos senhores será lembrado todos os dias em minha prática profissional.

“Os grandes feitos são conseguidos não pela força, mas pela perseverança.”

(Samuel Johnson)

“O homem não teria alcançado o possível se, repetidas vezes, não tivesse tentado o impossível.”

(Max Weber)

“Não há nada tão equitativamente distribuído no mundo como a inteligência: todos estão convencidos de que têm o suficiente.”

(René Descartes).

RESUMO

Chatterbots são aplicações que se propõem a conversar em linguagem natural, com o objetivo de simular o comportamento humano, causando a impressão de que o diálogo ocorre entre seres humanos e não entre um ser humano e uma máquina.
O Processamento de Linguagem Natural estuda a aproximação do homem e da máquina numa interação mais natural, e consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressas em linguagem natural.
Este trabalho de graduação tem como objetivo, apresentar o desenvolvimento de um *chatterbot*, designado Systembot, que irá interagir com alunos da Universidade Federal do Piauí, CSHNB, podendo ser utilizado para fornecer informações referentes ao curso de Sistemas de Informação, de maneira interativa, em forma de uma conversação. A classificação do *bot* é semelhante à de um sistema de FAQ, podendo, portanto ser classificado como um *FaqBot* (Podem ser utilizados para os mais diversos fins como agentes de entretenimento, agentes de auxílio ao aprendizado, e suporte ao usuário).

Palavras-chave: Inteligência Artificial**,** Agentes de conversação, Interação humano-computador.

ABSTRACT

Chatterbots are applications that propose a communication in natural language with the aim of simulating human behavior, causing the meaning that the dialogue takes place between human beings and not between a human and a machine.
The Natural Language Processing focuses on the approach of man and machine in a more natural interaction, and consists in the development of computational models for tasks that depend on information expressed in natural language.
This graduate work aims, the development of a *chatterbot*, nominated as Systembot, which has as a purpose to interact with students of the Federal University of Piauí - UFPI, CSHNB, which can be used by several courses, promoting a dynamic method that allows to stimulate the learning of the focused subject in a interactive way, in the form of a conversation. The classification of this bot_1 is similar to that one like a system of FAQ and can therefore be classified as a FaqBot (and can be used for various purposes such as entertainment agents, agents to aid learning, and user support).

Keywords: Artificial Intelligence, conversational Agents, human-computer Interaction.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIML	*Artificial Intelligence Markup Language*.
ALICE	*Artificial Linguistic Internet Computer Entity*.
Bot	Abreviação do Termo *Chatterbot*.
CHAT	*Conversational Hypertext Access Technology*.
FAQs	*Frequent Asked Questions*
HTML	*HiperText Markup Language*
JFRED	*Java Functional Responses Emulation Devices*
JRL	*JFRED Ruleset Language*
IA	*Artificial Intelligence*
PHP	*Hypertext Preprocessor*
PLN	Processamento de Linguagem Natural.
PyAIML	Interpretador para AIML, implementado em *Python*
SETL	*Set Theory and Mathematical Logic*
SGML	*Standard Generalized Markup Language*
XML	*Extensible Markup Language*
XML – R	*Extensible Markup Language – Reader*

Sumário

1 INTRODUÇÃO

O computador tornou-se cada vez mais presente no nosso cotidiano, realizando os mais variados tipos de tarefas, sendo que algumas dessas atividades envolvem “raciocínio”, podendo comportar-se como uma “extensão” da mente humana. A realização de tarefas desse propósito é um dos objetivos da Inteligência Artificial.

Os primeiros *chatterbots* construídos tinham como finalidade fazer uma imitação bem próxima do comportamento de um ser humano em um diálogo, de maneira que não fosse possível ao usuário do computador identificar quando estava interagindo (via teclado) com um computador ou outro homem. Esse tipo de programa foi idealizado por Alan Turing, em 1950.

Em seu artigo “*Can machines think*?”, propôs um teste, o Jogo da Imitação (*The Imitation Game*), conhecido como Teste de Turing, cujo objetivo era determinar se uma máquina pode pensar.

> “Para fazer isto é apresentada uma definição procedimental de inteligência de acordo com a seguinte estratégia. O jogo consiste em uma pessoa em um terminal (A) se comunicando com outro terminal (B), fisicamente separado de A, isto é, as pessoas não se veem. Caso a pessoa em A, após um determinado tempo de conversação, não consiga determinar se B está sendo controlado por outra pessoa ou um programa, o software pode ser considerado inteligente (Turing, 1950).”

A proposta desse jogo de imitação era caracterizar e estudar possibilidades relevantes de desenvolver uma máquina com a capacidade de raciocinar e ter um diálogo com um ser humano.

Este trabalho descreve o projeto e a construção de um *Chatterbot*, que possa auxiliar e interagir com alunos do Curso de Bacharelado em Sistemas de Informação, CSHNB, visando fornecer informações referentes à estrutura do curso, coordenação e informações importantes que devem ser fornecidas aos acadêmicos desse curso. Essas informações serão dispostas em forma de diálogo entre a máquina e os alunos através de um ambiente virtual baseado em uma aparência padrão de uma sala de chat.

Para o melhor desenvolvimento desse *chatterbot*, foram utilizadas ferramentas como o *Notepad++* e *Python Gui* para implementação dos códigos na linguagem Python e

AIML respectivamente. O objetivo da interface é proporcionar maior facilidade aos usuários em realizar uma conversação com a máquina, baseados no padrão de Humano-Computador que é obtido através do início de diálogo realizado pelo usuário, através a inserção de sentenças que serão digitadas na interface da aplicação e que após serem analisadas na base de conhecimento do bot, fornecerão a resposta mais indicada baseada no padrão da sentença de entrada.

A construção de aplicações desse porte é importante para a disponibilização da informação, de maneira simples e dinâmica, através de um diálogo em um ambiente virtual, e pode ser utilizado no dia a dia, em situações distintas, como atendimento virtual, ensino à distância, *FAQs* e consulta de conhecimento.

1.1 Objetivos

O principal objetivo deste trabalho é o desenvolvimento de um protótipo de um *chatterbot*, que possa interagir e fornecer informações referentes ao curso de Bacharelado em Sistemas de Informação (CSHNB), aos alunos do curso deste devido curso.

Para a construção desse *bot*, adotou-se uma abordagem bem simples, que consiste em proporcionar maior disponibilidade na informação aos alunos deste campus, promovendo uma melhor acessibilidade, e qualidade nas respostas emitidas pelo robô de conversação.

1.2 Organização do Trabalho

O documento está dividido em mais cinco capítulos.

No Capítulo 2, será feita uma abordagem sobre o conteúdo referente aos conceitos sobre Inteligência Artificial e, Processamento da Linguagem Natural.

No Capítulo 3, serão apresentados aspectos referentes a definição do termo *Chatterbot*, sua evolução histórica, gerações, principais tipos uso e as tecnologias usadas para a construção desses protótipos.

No Capitulo 4, será mostrada uma breve apresentação sobre o trabalho similar à aplicação desenvolvida, abordando suas principais contribuições, que serviram de auxílio para construção do protótipo.

No Capítulo 5, será descrito o protótipo desenvolvido, onde estão inseridas as etapas de construção, que se estendem desde a proposta inicial até o funcionamento do protótipo, além da implementação da interface e conversão dos códigos em Python através do PyMc.

No Capítulo 6, é apontada a conclusão do trabalho desenvolvido, sugestões sobre trabalhos futuros bem como a continuação no desenvolvimento do protótipo em questão.

2 Inteligência Artificial

Neste capitulo, iremos abordar um conteúdo referente, à definição da IA (*Artificial Inteligence* - Inteligência Artificial) e do Processamento da Linguagem Natural.

2.1. Uma breve abordagem sobre IA

Inteligência Artificial é o ramo da Ciência da Computação que se baseia no estudo da automação de comportamento inteligente de uma máquina.

> "Inteligência Artificial trata-se de um tipo de inteligência produzida pelo homem para dotar as máquinas de algum tipo de habilidade que simula a inteligência do homem (GALVÃO, 2003).”

Quando o termo Inteligência Artificial surgiu, em meados 1955, estimulou a expectativa de muitas pessoas sobre o fato de que as maquinas fossem capazes de possuir inteligência, e em outras, a dúvidas sobre o alcance dessa inteligência.

> "A origem da Inteligência é a história da lenta conscientização dos pesquisadores da área a respeito das limitações dos métodos computacionais, e da incompletude de conhecimentos a respeito dos mecanismos que tornam a inteligência possível (SGANDERLA, 2003).”

Mudanças das metas buscadas pela IA, são baseadas na construção de uma inteligência de caráter geral comparável à do ser humano, até os mais modestos aspectos atuais de tornar os computadores mais úteis através de ferramentas que auxiliam as atividades intelectuais de seres humanos.

No início dos anos setenta a IA oferecia técnicas robustas e complexas que permitiam o desenvolvimento de aplicações práticas envolvendo sistemas inteligentes. As pesquisas em IA estão relacionadas às diversas ciências e possuem diferentes enfoques. Qualquer que seja o enfoque, essa área tenta buscar resolver este problema através de agentes inteligentes. Esses

sistemas são criticados, uma vez que falham em questões que fogem de seu paradigma inicial. Tem-se como solução injetar mais conhecimento no sistema, ampliando suas capacidades (GALVÃO, 2003).

Por ser um tópico muito amplo, a Inteligência Artificial também está relacionada com psicologia, biologia, lógica matemática, linguística, engenharia, filosofia, entre outras áreas científicas, conforme mostra a Figura 1.

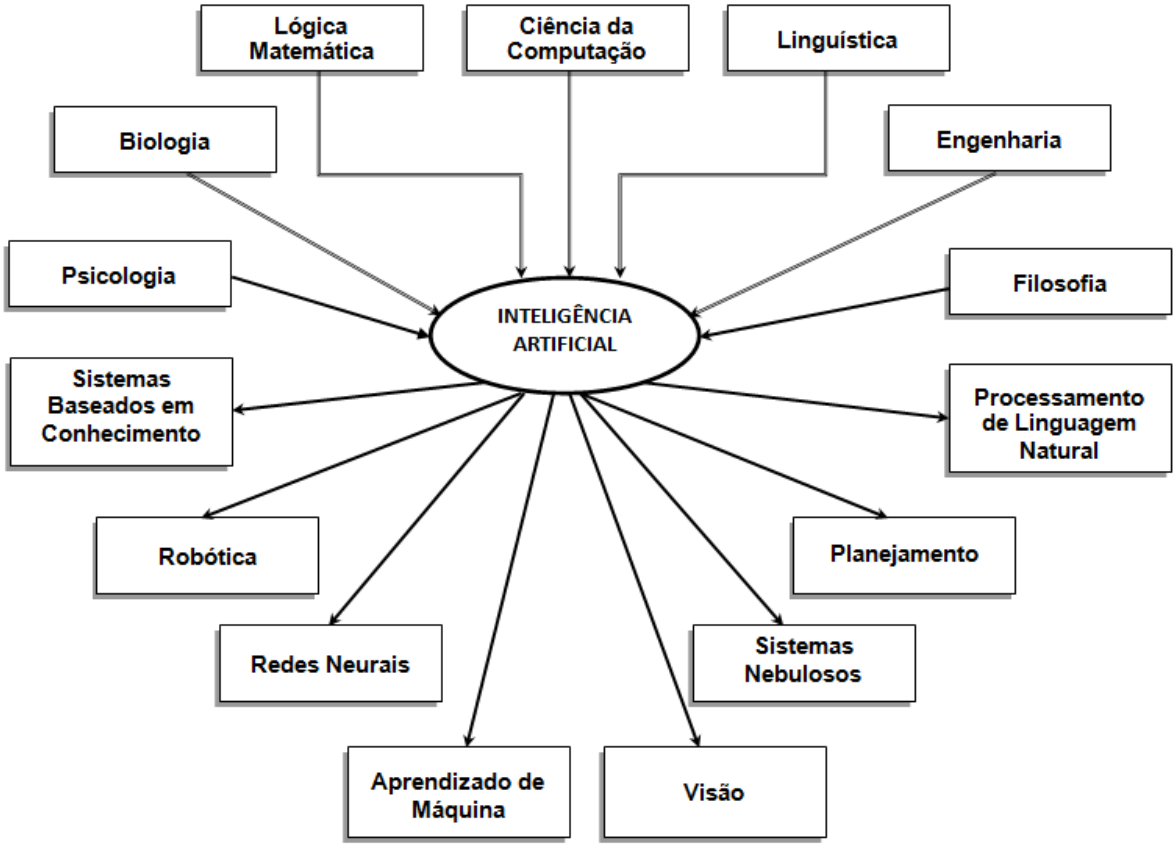

Figura 1 - Áreas relacionadas com a Inteligência Artificial.
Fonte: (GALVÃO, 2003).

Com base nessa Figura, observa-se que as setas que apontam para o termo Inteligência Artificial, representam áreas que passaram a aderir a essa área e através dessa junção, derivaram-se novas áreas que resultaram desse produto de interação, entre esses aspectos.

A inteligência Artificial está enquadrada nesse ramo de pesquisa, referente à implementação de um programa de computador, capaz de simular o comportamento de interação com um ser humano, fazendo com que o interlocutor imagine está falando com outro humano (LEONHARDT, 2005).

2.2. Processamento da Linguagem Natural (PLN)

O processamento de linguagem natural consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressas em alguma língua natural. As pesquisas em PLN estão voltadas a três aspectos da comunicação em língua natural (PEREIRA, 2007):

- Som: fonologia;
- Estrutura: morfologia e sintaxe;
- Significado: semântica e pragmática.

Nesse contexto, a fonologia está relacionada ao reconhecimento dos sons que compõem as palavras de uma língua. A morfologia reconhece as palavras em termos das unidades primitivas que a compõem. A sintaxe define a estrutura de uma frase, com base na forma como as palavras se relacionam nessa frase. A semântica associa significado a uma estrutura sintática, em termos dos significados das palavras que a compõem.

Finalmente, a pragmática verifica se o significado associado a uma estrutura sintática é realmente o significado mais apropriado no contexto considerado. Stairs e Reynolds (2006), afirmam que o processamento de linguagem natural permite que o computador reconheça comandos de voz em uma linguagem natural, e que existem três níveis que permitem esse reconhecimento que são:

- Comandos: Responsáveis pelo reconhecimento das palavras,
- Discreto: Reconhecimento da fala dita e as pausas entre as palavras,
- Contínuo: Reconhecimento da fala natural.

Este processamento de linguagem natural pode ser usado para recuperação de informações, sem a necessidade de utilização de busca por comandos ou palavras-chave. Pode-se falar em um microfone conectado ao computador e o computador converte a fala em arquivos de textos ou comandos. Os sistemas simples conseguem associar uma palavra digitada a uma palavra falada pelo microfone, sistemas mais avançados não precisam gravar as palavras.

Segundo Pereira (2007), os principais componentes presentes (Figura 2) no processamento da linguagem natural são responsáveis pela maneira como a linguagem é utilizada para comunicação como os significados obtidos na análise semântica agem sobre as

pessoas e seu contexto.

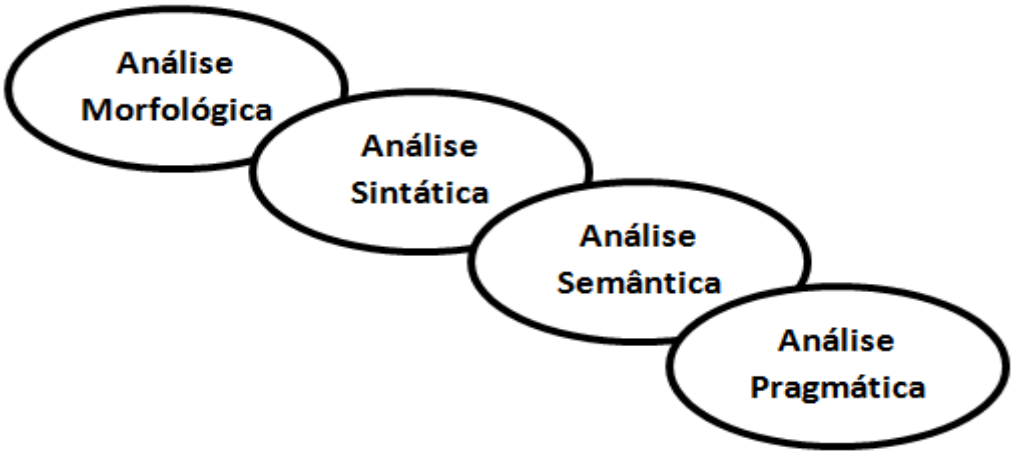

Figura 2 - Principais componentes do Processamento de Linguagem Natural.
Fonte: (PEREIRA, 2007).

2.2.1 Morfologia e Sintaxe

A análise morfológica é responsável pela identificação de um conjunto de palavras presentes numa sentença, que utilizam delimitadores, como espaços em branco ou caracteres especiais, representados por sinais de pontuação. Dessa forma, ao utilizar uma palavra em uma determinada sequência, essa palavra pode ser substituída por outra palavra que foi empregada no mesmo contexto e que melhor se adeque ao diálogo em questão, configurando uma sentença ainda válida. Dentro de um mesmo tipo de palavra, existem grupos de regras que caracterizam o comportamento de um subconjunto de vocábulos da linguagem.

Uma maneira mais simples de definir essas substituições seria especificar um conjunto de símbolos terminais, denotando palavras da linguagem, um conjunto de símbolos não terminais, denotando os componentes das sentenças, e um conjunto de regras de produção, que expandem símbolos não terminais numa sequência de símbolos terminais e não terminais (RICH, 1995). Dessa forma, a morfologia faz o tratamento das palavras de acordo com sua estrutura, forma, flexão e classificação, referindo-se a cada um dos tipos de classificação das palavras.

Outra análise importante nesse processo é a análise sintática que é capaz de responder se a sentença está correta ou não de no contexto gramatical. Essa análise busca definir qual a

estrutura de uma frase, baseando-se na forma como as palavras se relacionam, e verificando a sentença a partir da sequência de *Tokens*[1] recebidos e da sua adequação com a gramática da linguagem. Através da gramática da linguagem a ser analisada procura-se construir árvores sintáticas (Figura 3) para cada sentença, mostrando como as palavras estão relacionadas entre si.

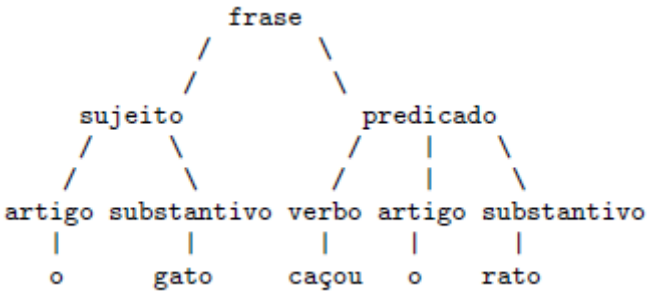

Figura 3 - Exemplo de uma frase analisada através de uma árvore sintática
Fonte: (PEREIRA, 2007).

2.2.2 Semântica e Pragmática

A análise semântica se refere ao significado das sentenças. É um ponto em que se é possível tratar as ocorrências de ambiguidades no contexto abordado e a diferenciação entre o significado e o sentido. A compreensão da relação entre as palavras é tão importante quanto à compreensão das próprias palavras.

Aspectos formais tentam descrever o sentido de uma frase, referente à tradução de sua estrutura sintática. Como não existe uma correspondência imediata entre sintaxe e semântica, uma mesma estrutura sintática pode dar origem a diferentes representações semânticas (OLIVEIRA, 2007).

A análise pragmática estuda a linguagem no contexto de sua utilização. É importante ressaltar que se deve fazer uma interpretação de todo o conteúdo e não mais analisar o significado de suas partes, do ponto de vista léxico e gramatical.

[1] *Tokens* são unidades básicas do texto do programa.

3 Os *Chatterbots*

O ser humano sempre buscou desenvolver técnicas para obter uma comunicação natural entre interlocutores. E isso vem sendo possível, através do crescente avanço da tecnologia, além da popularização do uso dos computadores pessoais, e da Internet. Vale ressaltar que esses aspectos proporcionaram a criação de tecnologias que viabilizassem a interação Humano-Computador. Essa interação acarretou o surgimento dos robôs de conversação, que são chamados de *Chatterbot*.

Os *chatterbot* ou agentes de conversação tem o objetivo de tornar amigável, a interação exercida pelo homem com a máquina, simulando a personalidade de um ser humano, tentando confundir o usuário que participa do diálogo.

3.1 O que é um *Chatterbot*?

O *chatterbot* é um sistema que utiliza linguagem natural para interagir com as pessoas, respondendo ou fornecendo informações, de maneira que se tenha a impressão de estar conversando com outra pessoa e não com um programa de computador (TEIXEIRA, 2005). Esses sistemas constituem uma das diversas maneiras de alcançar a humanização da máquina fornecida através da Inteligência Artificial. Uma definição mais completa sobre *chatterbot*, afirma que:

> *"Chatterbot* são sistemas que foram elaborados com o intuito de tornar mais familiar a interação entre o homem e os computadores, dando a impressão de que o sistema possui personalidade própria, e que se propõem a conversar em linguagem natural simulando o comportamento de um humano em uma conversação e que como objetivo interagir com o usuário respondendo a perguntas e estabelecendo um diálogo bem próximo do real, proporcionando entendimento do interlocutor. Sistemas desse porte têm a capacidade de explorar o comportamento social do usuário diante o computador até mesmo quando não são programados com essa intenção (SGANDERLA, 2003)."

A origem do termo vem da palavra Chat (*Conversational Hypertext Access Technology* - Hipertexto de Conversação de Acesso à Tecnologia), e a palavra Bot que tem sua origem na abreviação da palavra tcheca *Robota*, que significa trabalho (LEONHARDT,

2005).

Os *chatterbots* Atualmente são utilizados por milhares de usuários na *Web*, principalmente em ambientes como salas de chat, servindo como ferramenta de suporte e auxílio ao cliente, estudantes em ambientes de ensino à distância, bem como fornecer respostas rápidas e objetivas às pessoas sobre um domínio especifico.

3.1.1 Arquitetura Genérica de um *Chatterbot*

A maioria dos *chatterbot* desenvolvidos até os dias de hoje, possuem alguns componentes em comum referentes à sua estrutura. Esses componentes são modelo do usuário, modelo do domínio, interface gráfica e um elemento de execução. Esses componentes podem ser vistos como módulos individuais que compõem a arquitetura genérica da aplicação desenvolvida.

No modelo do usuário, temos a representação da base de conhecimento, que contém informações sobre os usuários do sistema. Este módulo é fundamental para o desenvolvimento mais natural do diálogo entre usuário e *chatterbot.*

O modelo do domínio representa a(s) base(s) de conhecimento, contendo as informações relacionadas aos temas pelo qual o *chatterbot* é capaz de conversar. Se *chatterbot* deseja conversar sobre vários assuntos, o sistema deve possuir várias bases de conhecimento, que serão selecionadas de acordo com a demanda. Esse tipo de seleção pode ser automática, baseado nas sentenças do diálogo com o usuário, ou manual, de acordo com alguma ação explícita do usuário, como por exemplo, uma seleção de um item disponível no menu.

A interface do *chatterbot* trata-se de um componente responsável pela forma de comunicação entre a aplicação e o usuário. Através dele, é possível que o usuário possa fazer requisições na forma de sentenças baseadas na linguagem natural para o *chatterbot*. Após esse processo de interação, a resposta é enviada ao usuário através da mesma interface, constituindo o diálogo entre o programa e o usuário.

O elemento de execução é o componente central de processamento, responsável por utiliza os modelos do usuário e domínio, para proporcionar uma conversação com o usuário mais natural. Este componente processa todas as requisições recebidas pelo componente interface (que recebe e emite informações visuais para o usuário), e aplica as técnicas e

estratégias para a escolha das respostas mais adequada, de acordo com as informações contidas no modelo do domínio. Na Figura 4, tem-se a representação gráfica da arquitetura genérica dos *chatterbot*.

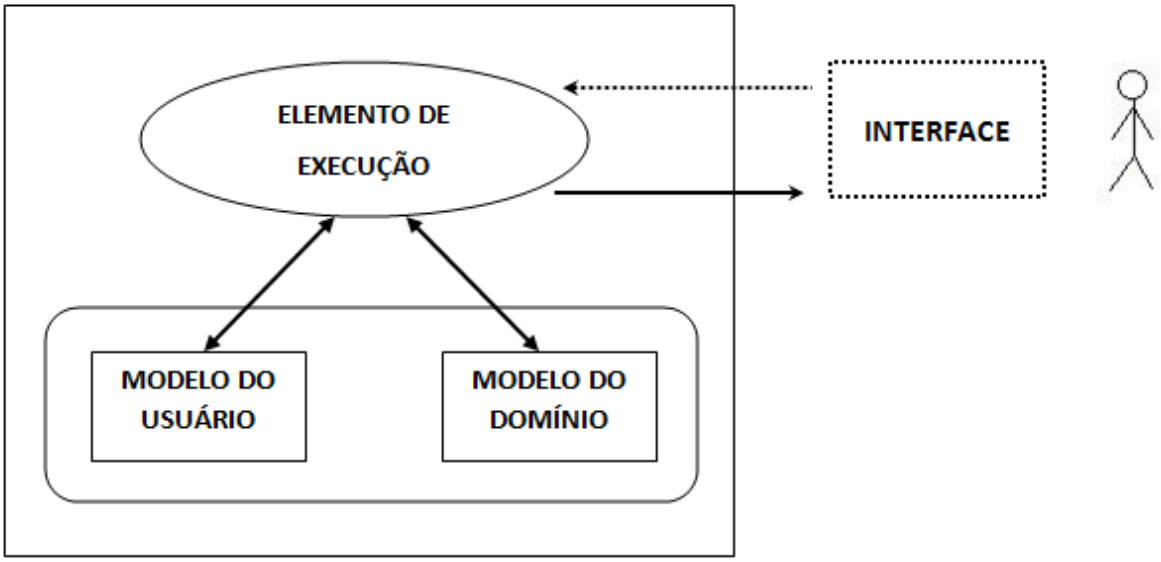

Figura 4 - Arquitetura Genérica para *Chatterbot*
Fonte: (LEONHARDT, 2005).

3.2 Evolução Histórica

Inicialmente, os computadores eram tidos como objetos que efetuavam somente operações aritméticas.

> "No ano de 1990, o jogo proposto por Alan Turing, ficou conhecido como Teste de Turing, e foi remodelado por um grupo de pesquisadores do *Cambridge Center for Behavioural Studies*, e foi o pontapé inicial, que promoveu um evento de grandeza internacional, denominado *Loebner Prize*. Esse evento tinha como objetivo, analisar e avaliar o desempenho dos *chatterbot*, através de diálogos feitos entre os jurados do evento e os sistemas de processamento em linguagem natural (NORVIG; RUSSEL, 2002)."

Com base na evolução, implementação e características dos *chatterbots*, destacam-se três gerações, que serão descritas a seguir.

3.2.1 Primeira Geração

Os *chatterbots* da primeira geração eram baseados em técnicas simples de casamento de padrão. Casamento de padrões é uma técnica em que é feito uma junção entre um conjunto de palavras-chave e um grupo de respostas relacionadas àquelas chaves, respeitando a ordem das chaves (CASSELL, 2003).

Um marco importante nessa geração foi à criação do *chatterbot* ELIZA, desenvolvido em 1966 por Joseph Weizenbaum, que possuía apenas 204 linhas de código fonte. Comportava-se como uma espécie de terapeuta, onde fazia com que o usuário revelasse suas informações pessoais, e teve como característica mais marcante, a forma de construção de determinadas frases, no qual o sistema tendia a repetir as frases dos pacientes, e em alguns casos modifica a frase do interlocutor, retornando uma pergunta relacionada ao que foi dito.

Apesar de ter sido uma das primeiras implementações, e ser bem simples, o *chatterbot* ELIZA possuía umas das personalidades mais definidas entre esses sistemas. Uma característica marcante em ELIZA é a maneira como ela modifica termos nas frases dos usuários, fazendo repetições mencionadas anteriormente pelo paciente, dando a entender que está entendendo a situação e transparecendo naturalidade, mostrando a habilidade de um terapeuta. Suas respostas eram baseadas nas buscas por palavras-chaves, e a saída era obtida por respostas pré-definidas de acordo com as palavras usadas pelo usuário, a partir de técnicas de casamento de padrões, além de sempre buscar uma palavra usada pelo ser humano para fazer uma nova pergunta (WEIZENBAUM, 1966).

Nessa geração, *chatterbot* como o *THE PC THERAPIST*, desenvolvido por Joseph Weintraub vencedor de quatro “Prêmios Loebner”, sendo três anos seguidos (1991, 1992 e 1993) e em 1995, e o HEX, desenvolvido por *Jason Hutchens*, vencedor desse mesmo prêmio no ano de 1996, são marcantes, principalmente por terem reconhecimento e utilizarem as técnicas utilizadas por ELIZA.

3.2.2 Segunda Geração

Na segunda geração de *chatterbot*, técnicas mais sofisticadas de Inteligência Artificial, passaram a serem utilizadas, as redes neurais. Um exemplo de *chatterbot* que utiliza essas tecnologias é JULIA. Julia foi desenvolvido em 1994 por Michael Mauldin na *Carnegie Mellon University*. Ela é um *TinyMUD* (multijogador em tempo real do mundo virtual), ou seja, um personagem dentro de um jogo de computador multi-usuários. Sua função é emular um jogador que auxilia os demais jogadores mapeando cavernas, indicando direções e os melhores caminhos a serem tomados, além de fornecer informações sobre outros jogadores, ambientes e objetos. Para isso, utiliza também no seu processo de decisão informações referentes a ações tomadas anteriormente (FONER, 1997).

JULIA destacou-se bastante por ser um agente que possui um modelo interno do mundo em sua base de conhecimento, tinha mecanismos que permitiam uma exploração autônoma, e possuía memória de estado, que permitia lembrar-se das últimas sentenças de entrada digitadas pelos usuários, podendo fornecer uma maior assistência, assim é capaz de interagir com o usuário de tal maneira que causa uma percepção de realismo mais efetiva em relação à apresentada por ELIZA. Em sua primeira versão apresentava um algoritmo bem simples, em que suas habilidades conversacionais utilizavam um sistema de casamento de padrões baseado em regras do tipo *IF-THEN-ELSE*. Esses padrões tinham mais de uma resposta, dentre as quais escolhia uma em forma aleatória.

Posteriormente, foi incluído um módulo baseado em redes neurais. Esse tipo de rede é composto por neurônios que contém os seguintes atributos: valor de ativação, conjunto de padrões, textos de resposta para o usuário, e aumento e inibição, que servem para ajustar o nível de ativação. Assim, quando o usuário digita uma mensagem e a envia, ela é comparada com todos os neurônios existentes, buscando casamento de padrões. Caso este ocorra, haverá uma mudança nos níveis de ativação.

O neurônio com o maior nível de ativação é o que determinará a resposta que será dada ao usuário. Por fim, através dos atributos aumento e inibição é feito um ajuste na rede (MAULDIN, 1999). Na Figura 5, é mostrado um exemplo de uma Rede neural.

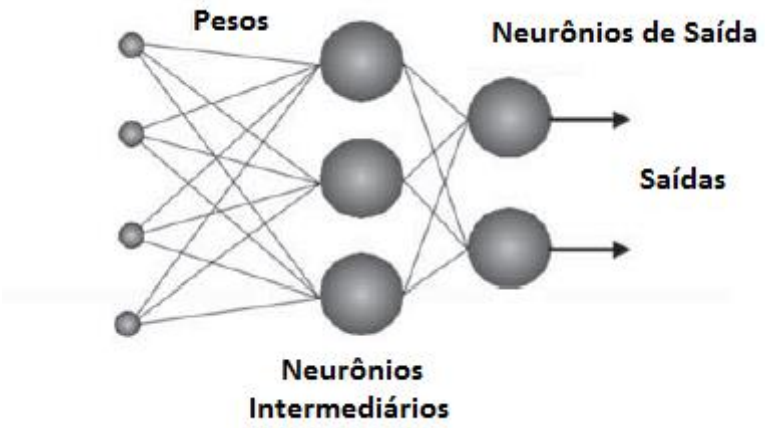

Figura 5 - Exemplo de uma Rede Neural
Fonte: (MAULDIN, 1999).

As habilidades de conversação dos sistemas dessa geração foram obtidas através de uma estruturação do módulo de aprendizado que possui uma rede de avaliação das respostas, e em cada nó da rede neural existem uns determinados tipos de padrões. Dessa forma, quando a frase emitida pelo usuário aciona um padrão, os nós que possuem esse modelo estimulam a ativação daquele setor da rede e envia a resposta ao usuário.

Outra aplicação importante nessa geração foi à criação de um framework voltado para o desenvolvimento de *chatterbot*, chamado JFRED (*Java Functional Responses Emulation Devices*).

JFRED é uma ferramenta desenvolvida através da linguagem Java, e é voltada para a construção de *chatterbot* para a *Web*. A base de conhecimento para *chatterbot* baseados nesse tipo de tecnologia é descrita a partir da utilização da linguagem JRL (*JFRED Ruleset Language*), baseada em regras que utilizam Lógica Fuzzy e um conjunto de palavras-chave para o casamento de padrões das sentenças de entrada fornecidas pelo usuário.

A ferramenta JFRED e o *chatterbot* ALBERT *One* são os destaques importantes dessa aplicação, e em conjunto foram os vencedores do Prêmio *Loebner* por dois anos consecutivos (1998 e 1999).

Mesmo que os *chatterbot* dessa geração tenham usado técnicas mais sofisticadas de Inteligência Artificial, não tiveram de certa forma os resultados esperados se comparados com os sistemas da primeira geração.

3.2.3 Terceira Geração

Os sistemas de terceira geração merecem destaque principalmente, por possuírem uma arquitetura mais complexa, em relação às gerações anteriores, e é marcada pela utilização da linguagem de Marcação AIML (*Artificial Intelligence Markup Language*), que é uma linguagem baseada em XML (*Extensible Markup Language*), utilizada para a elaboração da base de conhecimento de *chatterbot.*

Linguagens de marcação tem como base uma metalinguagem mais conhecida como SGML (*Standard Generalized Markup Language*). No Prêmio Loebner de 2000, o Dr. Richard Wallace apresentou o *chatterbot* ALICE (*Artificial Linguistic Internet Computer Entity*) que teve como principal inovação a utilização da AIML, e apesar de ser simples, obteve resultados melhores do que os *Chatterbot* anteriores.

A base de conhecimento de ALICE possui um vocabulário com mais de 5.000 palavras, além de um esquema que permite classificar uma determinada conversa com um ser humano, por idade, gênero, profissão entre outros. É um sistema desenvolvido em linguagem natural, e possui a capacidade de armazenar o nome do usuário, a conversação e o registro armazenado em logs, dessa maneira esse sistema segue o padrão de conversa e não tende a mudar de assunto aleatoriamente seguindo o padrão que foi iniciado referente à entrada do usuário.

ALICE possui uma base de conhecimento com aproximadamente 25.000 categorias distintas e tem um sistema utilizado para seleção de resposta, escrito em JAVA, que percorre toda a base de conhecimento, à procura de uma categoria que seja compatível com a sentença emitida pelo usuário. Essa busca analisa o padrão de entrada e quando encontra uma sentença válida retorna ao usuário a resposta mais convincente encontrada em sua base.

Com a utilização do AIML, podem-se ser feitas definições de múltiplas respostas para uma única pergunta e até implementar condições para a escolha de uma resposta. Com isso, o *chatterbot* torna-se mais abrangente e flexível em suas respostas (LEONHARDT, 2005, p. 5). Assim, o *chatterbot* faz a consulta da resposta para a pergunta na base de conhecimento; esse processo é feito pela máquina de inferência.

A máquina de inferência é responsável pela tradução da escrita humana para um entendimento do *chatterbot*, identificando e retornando com entendimento à altura. A primeira fase consiste em identificar caracteres especiais ou abreviações inseridas no texto e substituí-los por outras que o *bot* identifica como padrão. A segunda consiste na separação de

assunto que o usuário entrou, considerando a pontuação. Por último, são removidas as pontuações e aplicado um efeito de *uppercase*, ou seja, o texto escrito é convertido em maiúsculo (LEONHARDT, 2005, p. 60). Na Figura 6 é mostrada a estrutura de ALICE.

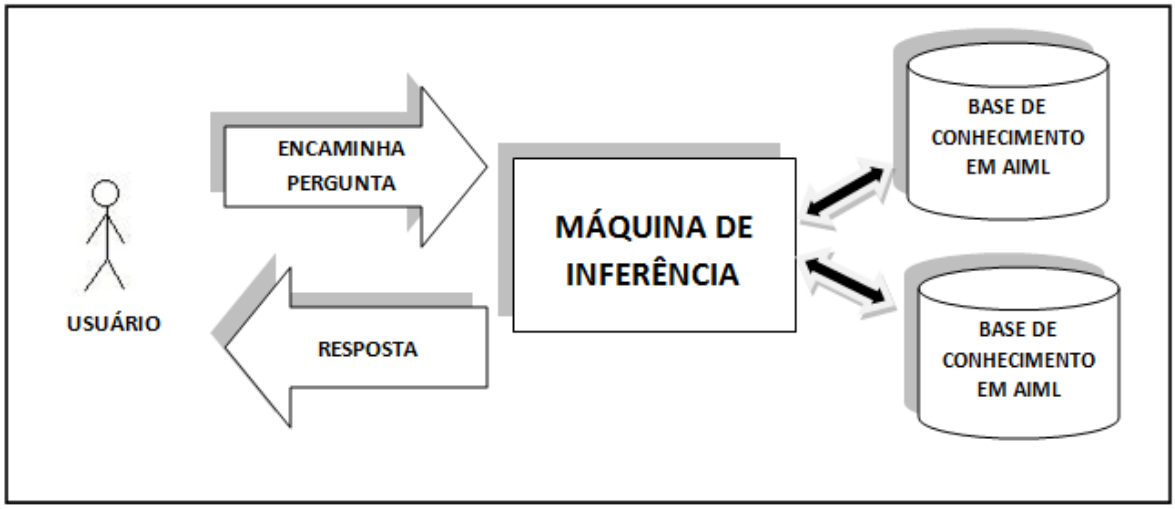

Figura 6 - Arquitetura de ALICE
Fonte: (LEONHARDT, 2005).

Analisando o esquema, e detalhando os passos temos:

1) A pessoa encaminha a pergunta ao *chatterbot*,

2) A máquina de inferência procura na base de conhecimento a resposta adequada para informar ao usuário,

3) Mostra essa resposta por meio de uma mensagem na tela do computador.

Para ser possível identificar a entrada do usuário na base de conhecimento e retornar uma resposta, utiliza-se o interpretador AIML. O interpretador é uma ferramenta *free*, desenvolvida na linguagem de programação Python; podemos definir o *Program* Y como a máquina de inferência. Esse programa possui uma ferramenta que mostra as perguntas dos usuários e as respostas dadas pelo *chatterbot*. Além disso, por intermédio dele é possível conFigurar diversos *chatterbot* rodando ao mesmo tempo (AIRES, 2008).

O AIML e ALICE representam ponto de partida para muitos outros projetos de *chatterbot* disponíveis hoje na Internet. Para isso, basta que seja desenvolvida uma nova base de conhecimentos em AIML. Atualmente existe uma fundação que promove a disseminação do software, que é gratuito e parte integrante do projeto GNU, bem como a construção outros *chatterbot* que se utilizem do sistema. A performance excepcional deste software no concurso de *Loebner* pode ser atribuída a três fatores principais (NORVIG; RUSSEL, 2002):

• A licença gratuita do software permite com que as pessoas estejam familiarizadas com o ambiente de desenvolvimento e possam adicionar funcionalidades que permitam ao bot se comunicar mais robustamente com usuários;
• A linguagem utilizada para armazenar o conhecimento do bot é de fácil aprendizagem e permite que conhecimentos sejam adicionados a qualquer momento, por qualquer usuário, de forma a representar seu próprio conhecimento e suas próprias respostas nas mesmas ocasiões;
• A plataforma ALICE e a linguagem AIML são independentes, ou seja, não é necessário ser um programador com grande conhecimento para utilizar o programa.

Este terceiro fator se complementa com o fato de que a plataforma não exige recursos extras para que seja possível sua utilização e de que existem diversas implementações do sistema (*Java*, *Perl*, *C++*, *PHP*), responsável pelo processamento de bases de conhecimento em AIML, sendo todas de código aberto. Além disso, a ênfase do trabalho de ALICE é no desenvolvimento da linguagem AIML, e não na tecnologia de suporte ou em uma aplicação particular do *chatterbot* (NEVES, 2005).

3.3 Tipos de Chatterbot e principais aplicações desenvolvidas

Entre os principais usos dessas aplicações, tem maior destaque, os *chatterbot* FaqBots, de Entretenimento, Ensino à Distância, FAQs (*Frequent Asked Questions*), Suporte ao consumidor e de Propósito Geral (FILHO, 2009).

3.3.1 *FaqBots*

FaqBots (*Frequent Asked Questions*) são encarregados de responder as perguntas dos usuários baseados em suas bases de FAQs (Questões respondidas com frequência), além de poderem ser utilizados tanto para prover informações sobre empresas, produtos e serviços, como em ambientes virtuais de ensino para responder dúvidas dos alunos. Esses *bots* podem ser usados, também, em ambientes virtuais de estudo, para responder a dúvidas de alunos

acerca do assunto sendo estudado.

FaqBots, não tem o propósito de simular um ser humano, muito menos tentar iniciar um diálogo relacionado que não seja relacionado à indagação do usuário. Diferente dos *chatterbot* de entretenimento, que foram desenvolvidos com o intuito de auxiliar as pessoas a obterem respostas de forma mais efetiva, rápida e simples sobre um domínio especifico. Nesse tipo de *bot*, quando não se encontra uma resposta adequada à pergunta do usuário, é preferível que este assuma ignorância, evitando assim, problemas que podem ser gerados, em caso de uma resposta inadequada.

Um exemplo de aplicação que se encaixa nessa classe é o *chatterbot Shallow Red* da Neuromedia, empresa de software que desenvolve *chatterbot* para funcionar na Internet. Ele guarda as últimas sentenças do diálogo com o usuário, e também é capaz de contextualizar algumas características do usuário. Este programa consegue conversar muito bem sobre o seu domínio, o qual consiste em características e condições de venda do *Neuroserver* (KRISTOL; MONTULLI, 1997).

Ele assume ignorância quando questionado sobre assuntos fora do seu conhecimento. A seleção de respostas é feita através de casamento de padrões: palavras-chave selecionadas a partir de cada questão são comparadas a uma lista interna de combinações de chaves que indexam as respostas.

3.3.2 Entretenimento

O Entretenimento é uma das aplicações mais comuns para os *Chatterbot*. Para um bom desempenho deve-se ter um comportamento dinâmico e capaz de dar sensação da existência de uma personalidade.

Dentro desta classe, encontram-se também alguns *chatterbot* criados com o objetivo inicial de estudar a complexidade na comunicação em linguagem natural entre homens e máquinas. O exemplo mais representativo deste tipo de programa é o *chatterbot* ELIZA (WEIZENBAUM, 1966).

Os *chatterbot* dessa classificação foram definidos, com o intuito de divertir os usuários que os acessam, de tentar estudar com mais precisão a complexidade na comunicação em linguagem natural entre homens e máquinas, e são sistemas desenvolvidos para que simulem

pessoas interagindo em salas de bate-papo. Entretanto é preciso que sejam dinâmicos e consequentemente simulem um comportamento de um ser humano.

Outro exemplo de aplicação inclusa nesse grupo é o ED, que se trata de um projeto mantido pela PETROBRAS que traz informações sobre o meio ambiente, conversando sobre uso ciente de energia e combustíveis. É possível esclarecer dúvidas sobre esses assuntos, simulando um diálogo, e consequentemente, agregar conhecimento de uma maneira mais rápida.

3.3.3 Ensino à Distância

Ensino à distância, busca obter e manter ativo, o estimulo ao aprendizado dos alunos que na maioria dos casos acaba tornando-se pouco flexível. O uso de *chatterbot* nesse ambiente possibilita os alunos pesquisar informações de seu interesse num diálogo bem interessante pela interatividade (GALVÃO, 2003).

Dependendo do seu potencial comunicativo do *chatterbot*, pode existir um tipo de interação mais humanizado, incrementando a tecnologia de interação. Os internautas demonstram grande interesse em interagir com esses robôs. Por ser uma novidade, esse fato proporciona atenção do público, atraindo-o a utilizar o sistema. O diálogo proporcionado pelo *chatterbot* pode tornar-se um incentivo ao trabalho do aluno, pois é necessária uma participação mais ativa. A integração do robô num ambiente virtual de aprendizagem interativo e dinâmico ainda pode ser enriquecida com animações, filmes, sons e chats com outras pessoas (RABUSKE, 1995).

O *chatterbot* ADELE tem sido utilizado, em sistemas de ensino à distância, executando tarefas como: responder questões dos estudantes, efetuar demonstrações, e monitorar as ações dos estudantes (SHAW; JOHNSON, 1997).

Outro exemplo é o robô Einstein, desenvolvido pela empresa Artificial Life para ensinar física. Caso o estudante não responda uma questão dirigida a ele por um longo tempo, ele é advertido para que passe a prestar atenção na aula (FORBES, 2000).

Chatterbot voltados ao Ensino à Distância devem ser projetados de forma que possam tomar atitudes perante os usuários como, por exemplo, repreender aqueles que não estejam se

empenhando nos estudos.

É importante identificar que comportamentos são apropriados para estimular a aprendizagem, caso contrário, é possível gerar distúrbios no comportamento do estudante como, por exemplo, ansiedade ou timidez (ANDRADE et. al. , 2002).

3.3.4 Suporte ao Consumidor

Serviços de atendimento ao consumidor são indispensáveis para o bom funcionamento de um negócio, porém o custo associado a esse tipo de serviço é alto. Estudos mostram que a maior parte das dúvidas dos usuários, podem ser respondidas, utilizando uma pequena parte base de conhecimento de suporte ao usuário (GIANFORTE, 2002).

Essa classificação de *chatterbot* é muito parecida com a dos FaqBots, e a diferença entre esses eles, é a necessidade de interação com o usuário. Podemos explicar essa diferença da seguinte maneira: Um cliente está reclamando do mau funcionamento de um produto fornecido pela empresa que mantém o *chatterbot*, então o bot realizará uma série de perguntas referentes ao produto, buscando promover uma ajuda eficaz ao cliente. Aplicações desse modelo são muito importantes para empresas e clientes, evitando excesso de comunicações que seriam desnecessárias caso fosse um problema simples de ser resolvido.

Um *chatterbot* com essa classificação é o *Hank*, um bot desenvolvido pela empresa Coca-Cola como o objetivo de esclarecer dúvidas sobre a empresa e seus produtos. Quando não é capaz de responder uma questão, *Hank* sugere ao usuário tentar refazer a questão ou entrar em contato diretamente com o suporte através de um link fornecido por ele (COCA-COLA, 2002).

3.3.5 Propósito Geral

Alguns modelos de *bots* vêm sendo classificados como *Chatterbots* de Propósito Geral, devido ao fato de não estarem inseridos em nenhuma das características mencionadas

acima, ou mesmo por possuir tais características presentes em mais de um tipo de classificação. O Ultra Hal, da *Zabaware*, está inserido nessa classificação, e vem sendo utilizado como ferramenta de auxílio no aprendizado de diversos assuntos que são recuperados de bancos de dados disponíveis na Internet.

Outra funcionalidade dessa aplicação é servir de interface de comandos em linguagem natural para o Windows, onde é possível pedir ao programa para executar um determinado aplicativo apenas mencionando isto em linguagem natural.

Outra aplicação relevante, voltada ao aspecto de interface é o CHAT (*Conversational Hypertext Access Technology*).

> "O CHAT é um programa que provê uma interface em linguagem natural para acesso fácil a documentos eletrônicos, e que esses documentos podem ser retornados em um formato multimídia, ou podem ser retornados apenas endereços de sites contendo informações interessantes sobre o tema. O formato da resposta vai depender da pergunta feita pelo usuário (WHALEN, 1996)."

O foco principal desse programa é analisar e avaliar a natureza do processo de comunicação homem-computador. Para que essa avaliação fosse realizada, foi necessário um estudo sobre o comportamento humano diante da máquina, de forma a descobrir o funcionamento da mente em interações através de linguagem natural.

O funcionamento dessa aplicação é baseado em padrões de mapeamentos das perguntas mais comuns de usuários nas respostas adequadas, entretanto não podemos considerar o CHAT como um *chatterbot*, mas devemos considerar que seu desenvolvimento impulsionou e colaborou a aceleração do desenvolvimento de pesquisas na área.

A *Robitron Software Research* desenvolveu uma tecnologia denominada JFRED (*Java Functional Responses Emulation Devices*), que permitia criar as chamadas personalidades automatizadas interativas, resumindo, um *chatterbot* com várias utilidades como: suporte técnico baseado em resposta às questões comuns, suporte ao consumidor e serviços de propaganda em *banners*. O mais interessante nesta tecnologia, não é o conjunto de aplicações que ela pode gerar, mas a sua estrutura e seu funcionamento, e possui o intuito de prover uma interface de software em linguagem natural com as seguintes características: independência de plataforma, servidor *multi-threaded*, devido a ser uma aplicação Java, utilização de lógica *fuzzy* baseado em regras de inteligência artificial, aprendizado baseado em frames e

independência de língua (GARNER; NATHAN, 1997).

3.4 Tecnologias usadas na construção de um *Chatterbot*

O desenvolvimento de um *chatterbot* consiste em um conjunto de práticas, conceitos e técnicas que possibilitam a sua execução, criando um ambiente virtual para a sua construção. O processo de construção é complicado, e deve procurar ser previsível quanto ao impasse de encontrar problemas na falta de uma base de conhecimento consistente, que possibilite um nível de interação ao bot. Para que seja possível obter uma resposta satisfatória, é necessário preencher a base de conhecimento do bot, de tal maneira que essa resposta esteja associada ao padrão de entrada fornecido pelo usuário (NORVIG; RUSSEL, 2002).

Nesta seção, serão descritos a Linguagem AIML, que atualmente é a principal ferramenta para o desenvolvimento de *chatterbot* e criação da sua base de conhecimento, a base de conhecimento AIML, as principais técnicas e estratégias de resposta, e os principais interpretadores, além de descrever o uso do *Program* Y, que é baseado na linguagem de programação *Python*. Essas ferramentas foram utilizadas no desenvolvimento do trabalho, que será descrito detalhadamente no capítulo 5.

3.4.1 Linguagem AIML

A linguagem AIML, baseada em XML, é descrita através de identificadores denominados tags. Essa linguagem foi desenvolvida tendo como inspiração o modelo de casamento de padrões proposto pelo *chatterbot* ELIZA. A construção de grande parte de *Chatterbot*, que atualmente possuem aceitação e desempenho em testes importantes, utilizam o AIML como base de conhecimento, o padrão foi desenvolvido por Dr. Wallace (WALLACE, 2007) e utilizado na implementação de ALICE que basicamente estrutura as informações de forma simples baseado no XML.

O padrão XML (*Extended Markup Language*) foi desenvolvido em 1998 pelo consórcio W3. Sua proposta inicial e principal motivação era a criação de uma estrutura em que fosse possível vincular informações relativas a ela, e ao significado dos dados, tornando-o autodescritivo, e ainda podendo categorizar os dados. Uma das principais *tags* de AIML é a

categoria “< *category* >”, que é composta por uma sentença enviada pelo usuário para o *chatterbot*, padrão de entrada “< *pattern* >”, a qual está associada a uma ou mais sentenças de resposta “< *template* >” (NEVES, 2005).

A facilidade em estruturar a informação fez com que para os projetos de *chatterbot* fosse criado um padrão baseado no XML, o conhecido AIML. O XML é, portanto, mais uma metalinguagem de marcação de texto. Formalmente pode-se dizer que é um conjunto de regras para definir *tags* semânticas que quebram um documento em partes e identifica diferentes partes do documento.

A criação da linguagem AIML, permitiu que os padrões da base de estímulo-resposta possam ser hospedados e processados pela Web, aumentando sua portabilidade. Sua ideia inicial era um sistema capaz de facilitar a implementação de um Bot baseado no ALICE, dentre os seus principais objetivos tem-se:

- Fácil aprendizado;
- Utilização de um conceito mínimo, necessário para permitir o funcionamento de um sistema de estímulo-resposta;
- Compatível com XML;
- Facilidade de escrever em AIML e documentar o processo;
- Os objetos devem ser de fácil compreensão humana;
- O projeto AIML deve ser formal e conciso.

O desenvolvimento do padrão foi iniciado por Dr. Richard Wallace e a comunidade ALICEBOT durante os anos de 1995 e 2000.

As estruturas de objetos AIML são semelhantes a XML, embora objetos XML possam conter documentos AIML. Entre os objetivos desta linguagem destacamos: AIML deve ser fácil para pessoas aprenderem; deve codificar o mínimo conceito necessário para poder modelar um conhecimento de estimulo-resposta; e AIML não deve incorporar dependências de qualquer outra linguagem (WALLACE, 2007).

É de fundamental importância realizar uma abordagem parte importantes dessa linguagem: a base de conhecimento e a estrutura da linguagem em questão, pois o AIML é responsável por realizar o trabalho de armazenar os diálogos e promover a comunicação e acesso da aplicação numa base de dados, e é necessário entender a estrutura dessa base em questão.

3.4.2 Base de Conhecimento AIML

A base de conhecimento, também conhecida como base de dados, é tida como um sistema responsável por mapear todo o conteúdo em uma coleção de dados, coletado nas informações gravadas nessa base. Essas informações são armazenadas de forma organizada e que permita realização de consultas e busca de conhecimento.

O conhecimento é como uma informação armazenada, ou os modelos usados pela pessoa ou máquina para interpretar, predizer e responder apropriadamente (FINNIN; FRITZSON, 1994). Tendo como base essa definição, foi observado que a base de conhecimento de um *chatterbot* é de fundamental importância, pois irá pré-determinar a qualidade e capacidade de reconhecimento de uma entrada fornecida pelo usuário e responder da maneira mais coerente possível, baseados no contexto da conversação em pauta.

Nos dias de hoje, a maioria dos *chatterbot* que possuem aceitação no mercado, utilizam AIML como base de conhecimento (WALLACE, 2007). Para a criação de um documento em AIML, devem-se respeitar as instruções e padrões adotados pelo XML, contribuindo para a construção de um arquivo AIML dentro dos padrões.

XML é uma metalinguagem de marcação de texto, com um conjunto de regras para definir tags semânticas que quebram um documento em partes e identifica diferentes partes de um documento.

Um documento XML é considerado bem formado se ele respeitar as seguintes situações:

- Reúnem-se todas as restrições definidas na especificação do XML (BRAY; MALER; PAOLI, 2000);
- Cada uma das entidades analisadas, que está relacionada no documento é bem formada;
- Contém um ou mais elementos;
- Elementos delimitados pelo início e fim de *tags*;
- Há exatamente um elemento, chamado de raiz, ou elemento do documento.

A estrutura presente na linguagem AIML é praticamente semelhante ao XML, seguindo os mesmos padrões de formatação, inserção de caracteres e comentários. O objeto AIML é formado de uma estrutura física, que é composta por categorias, e lógica, composta por elementos ou referencias de caracteres especiais indicados em uma devida marcação. A versão atual da linguagem AIML é 1.0.1 e suporta vários caracteres em um padrão de entrada. A estrutura do documento AIML se inicia com a declaração que informa a versão em questão

do XML, conforme a Figura 7.

```
<?xml version="1.0" encoding="UTF-8"?>
        <aiml version="1.0">
        ...
        </aiml>
```

Figura 7 - Estrutura de um Documento AIML

O AIML descreve uma classe de objetos de dados chamados de objetos AIML, que são constituídos de unidades denominadas categorias "<*category*>", e também é possível descrever o comportamento que o *bot* deverá assumir.

Cada categoria é representada por uma tag <*category*> é constituída por um componente de decomposição da sentença digitada, que chamamos de *pattern*, pelo usuário e por um componente de análise e construção da sentença devolvida pelo bot ao usuário, denominado *template*. Na Figura 8, temos o exemplo da sintaxe de uma categoria AIML.

```
<category>
        <pattern> COMO VAI VOCE?</pattern>
        <template> Eu vou bem. Obrigado!</template>
</category>
```

Figura 8 - Exemplo do uso da sintaxe básica de uma categoria em AIML

Ao utilizar um sistema para casamento de padrões simples, AIML usa dois caracteres especiais que criam um ponto de referência no padrão de entrada para apontar os termos da decomposição, que são o asterisco ("*") e a *underline* ("_") sendo que este caractere tem prioridade maior que o anterior, que tem o mesmo valor atribuído ao símbolo 0 (zero), representando um termo composto por uma sequência infinita de palavras.

Para a recuperação do conteúdo da sentença que casou com o os termos capturados pelos caracteres especiais podem ser referenciados através da tag <*star index*="i"/>, onde i representa o i-*ésimo* termo capturado por um caractere especial.

A sintaxe de uma categoria em AIML com o uso desses caracteres e a tag star index são mostrados na Figura 9.

```
<category>
        <pattern>*OI* TUDO BEM </pattern>
        <template> VOU BEM, OBRIGADO E VOCE?
        <star index="1"> </template>
</category>
```

Figura 9 - Sintaxe básica de uma categoria em AIML, com uso do caractere especial asterisco "*"

Em AIML utilizamos *tags* para criação de sentenças que em conjunto com o interpretador, analisam o padrão de entrada e a partir desse padrão buscam responder o usuário da maneira mais coerente possível. No início dessa seção falamos sobre *category*, *template*, *pattern* e star. Iremos agora abordar outros componentes que também são fundamentais no processo de interação mais dinâmica e transparente ao usuário.

a) Tags Set e Get

AIML define duas *tags*, usadas para o armazenamento e recuperação de variáveis da memória do *chatterbot*. São elas, as *tags set* e *get*. Na Figura 10 e 11, temos a sintaxe de uso e um exemplo de uso desses componentes.

```
<set name="variavelX">Valor de Entrada </set>
<get name="variavelX"/>
```

Figura 10 - Sintaxe de uso das *tags set* e *get*.

```
<category>
        <pattern>OLA, MEU NOME E *</pattern>
          <template>Ola, Muito Prazer em conhecer voce <set name="name">
<star/>
</set>
</template>
</category>
<category>
        <pattern>VOCE LEMBRA DO MEU NOME</pattern>
<template>Lembro sim. Seu nome e Junior, estou certo? <get name="name"/>.
</template>
</category>

RESULTADO DA CONVERSAÇÃO
-------------------------------------------------------
Usuário > Ola, meu nome e Junior.
Systembot > Ola, Muito Prazer em conhecer voce Junior.
Usuário > Voce lembra do meu nome.
Systembot>  Lembro sim. Seu nome e Junior, estou certo?
-------------------------------------------------------
```

Figura 11 - Exemplo de um diálogo, usando as *tags set* e *get.*

b) Tag Think

A *tag* <*think*> tem como função realizar o processamento de informações sem exibir nas respostas do diálogo, modificando as tags que ela contém determinando que, após o seu processamento, o resultado não deve ser retornado. Esse mecanismo simula a ideia de que o *chatterbot* está pensando.

Na Figura 12 e 13, temos exemplos da sintaxe de uso dessa *tag.*

```
<think> Conteúdo a ser processado </think>
```

Figura 12 - Sintaxe de uso da *tag think*.

```
<category>
        <pattern> VOCE E DIFICIL</pattern>
        <template> Eu estou tentando ser tao simples quanto possível <think><set name="it"></set></think>
        </template>
</category>
```

Figura 13 - Exemplo de uso da tag think.

Essa *tag* realiza o processamento de informações sem exibir nas respostas do diálogo e serve para que o interpretador processe informações sem retornar nada ao usuário do *chatterbot*. Utilizar tópicos é uma forma de diferenciar o tema da conversação e para casos

onde é necessário utilizar o mesmo padrão de entrada e retornar saídas diferentes.

c) Tags that e topic

AIML permite abordar o conteúdo ou mesmo situar tal fato no tempo e no espaço, do universo em que está envolvido, em um diálogo. Para isso, ela introduz as *tags that e topic*. A *tag* <that>, possui a sintaxe semelhante à *tag* <pattern>, utilizando os mesmos caracteres especiais. Ela prevê o tratamento de repetições de sentenças, comparando sua forma. Essa *tag* será ativada se o texto da conversa anterior contiver o conteúdo referenciado na *tag* that. Quando esta *tag* encontra-se dentro de uma categoria, é necessário fazer o casamento de padrão definido na tag *pattern*, deve-se verificar se a última frase dita pelo robô com a *tag* <that>. Sua sintaxe de uso e um exemplo de seu uso são mostrados respectivamente nas Figuras 14 e 15.

```
<that> ENTRADA </that>
```

Figura 14 - Sintaxe de uso da *tag that*

```
<category>
    <pattern>VAMOS CONVERSAR. ME PERGUNTE ALGO.</pattern>
        <template>VOCE GOSTA DE INFORMATICA</template>
</category>

<category>
    <pattern> GOSTO BASTANTE </pattern>
        <that> VOCE GOSTA DE INFORMATICA </that>
        <template> Que legal, Eu tambem gosto muito de assuntos sobre informatica </template>
```

Figura 15 - Exemplo de uso da *tag that.*

Quando a *tag that* é utilizada em uma categoria, essa categoria só será ativada se o texto de uma determinada sentença anterior contiver o conteúdo referenciado na *tag*. Esse é um recurso opcional para manter o contexto da conversação, e pode ser utilizado no caso do *chatterbot* pedir algo ou fazer alguma pergunta.

Em uma conversação é comum que as pessoas falem sobre diversos tópicos. Para permitir que o bot possa aprimorar cada vez mais o processo de escolha da resposta para o usuário, é necessário modelar a base a partir de tópicos.

O tópico em questão é determinado através da *tag* <*set*> utilizando uma variável chamada *topic*. Na Figura 16, temos a sintaxe de uso dessa *tag*.

```
<topic name="nome">
--- Categorias ----
</topic>
```

Figura 16 - Sintaxe de Uso da *tag topic*.

Exemplificando o uso dessa *tag*, teremos como base o seguinte exemplo: Em determinada situação, o bot não foi programado para responder determinada pergunta, ou não encontrou a resposta especifica coerente com o padrão de entrada, porém sabe qual assunto que está sendo focado. Com o uso dessa *tag*, é possível devolver uma resposta que pertença ao contexto do tópico, melhorando assim a qualidade do diálogo.

A *tag topic* é inserida antes da *tag category*, podendo ser incluídas dentro de um tópico várias *tags category*. Após isso, deve-se definir qual tópico está sendo utilizado através da tag set. Na Figura 17, é mostrado um trecho de um diálogo entre o bot e o usuário com o uso da *tag topic*

```
<category>
        <pattern> SOU SEU MELHOR AMIGO</pattern>
        <template> Eu nao gosto de escolher favoritos. <think>
                <set name="it"><set name="topic"> SOU SEU MELHOR AMIGO </set></set>
                </think>
        </template>
</category>

RESULTADO DO DIÁLOGO
----------------------------------------
Usuário > Sou seu melhor amigo.
SystemBot> Eu nao gosto de escolher favoritos.
----------------------------------------
```

Figura 17 - Exemplo de uso da *tag topic*.

A tag topic é um elemento opcional de alto nível que contêm tags categoria, e por sua vez tem um atributo de nome necessário que deve conter uma expressão de padrão simples, e permite conter um ou mais elementos na categoria. Utilizar tópicos é uma maneira de diferenciar o tema da conversação e para casos onde é necessário utilizar o mesmo padrão de entrada e retornar saídas diferentes.

d) Tags srai e condition

A *tag* <srai> indica que o sistema de seleção de regras deve avaliar o seu conteúdo como se fosse uma sentença digitada pelo usuário. Essa tag possibilita uma forma recursiva de simplificação da estrutura de entrada. Então, além de permitir que diferentes padrões de entrada sejam direcionados a uma mesma resposta, implementa-se um mecanismo semelhante à chamada de procedimento, comum em linguagens de programação.

Sua necessidade surgiu dos múltiplos padrões de pergunta que tem a mesma resposta. Nas Figuras 18 e 19, são ilustrados respectivamente a sintaxe de uso dessa tag, e um exemplo de uso.

```
<srai> Conteúdo de ENTRADA </srai>
```

Figura 18 - Sintaxe de uso da *tag srai.*

```
<category>
        <pattern>OLA</pattern>
        <template>
                <srai>OI</srai>
        </template>
</category>
<category>
        <pattern>OLA ROBO</pattern>
        <template>
                <srai>OI</srai>
        </template>
</category>

RESULTADO DO DIÁLOGO
---------------------------------------
Usuário> Ola
Systembot> Oi
Usuário> Ola robo
Systembot> Oi
---------------------------------------
```

Figura 19 - Exemplo do uso da *tag srai*.

A linguagem AIML possui uma tag específica, para simular condições, no qual é informada uma variável, um valor de critério e então um retorno, semelhante às estruturas condicionais das diversas linguagens existentes. A *tag condition* é utilizada dentro da *tag template*, também é possível colocar condições dentro de condições. Na Figura 20, é mostrado um exemplo de uso dessa *tag*.

```
<category>
      <pattern>OI</pattern>
      <template>
            <condition name="session" value="saudarbot">
                  <condition name="emotion" value="calmo">
                        Nao nos saudamos anteriormente?
                  </condition>
                  <condition name="emotion" value="xingarbot">
               Essa conversa esta sem graca e repetitiva.
            </condition>
             </condition>
      </template>
</category>
```

Figura 20 - Uso da *tag condition* em uma categoria AIML

e) *Tags li (List-condition)* e *random*

Na *List-condition tag*, é aberta uma *tag condition* e os critérios são distribuídos dentro de *tags li*. Na Figura 21 é apresentado um exemplo, onde a *tag li* que não contem atributo e valor de critério é considerado como valor padrão, default, no caso de nenhuma das outras condições terem sido atendidas (CORRÊA, 2001).

```
<category>
      <pattern>OI</pattern>
      <template>
            <condition>
                  <li name="session" value="saudarbot">
                        Nao nos saudamos anteriormente
                  </li>
                  <li name="session" value="sair">
                  Foi mal, estava de saida mesmo.
                  </li>
                  <li>
                  Ola, tudo bem?
                  </li>
            </condition>
      </template>
</category>
```

Figura 21 - Exemplo de uso da *List-Condition Tag*

A *tag random* informa ao interpretador que ele deve escolher uma resposta aleatória dentre as possíveis respostas que são separadas com a tag li. Na Figura 22, temos um trecho de uma categoria que utiliza essas *tags*.

```
category>
      <pattern>OI</pattern>
        <template>
            <random>
             <li> Ola </li>
             <li> Ola,Tudo bem <random>
             <li> Ola, Tudo tranquilo </li>
            </random>
        </template>
</category>
```

Figura 22 - Exemplo de uso das *tags li* e *random*.

f) *Tags* de Substituição

Em AIML é possível realizar substituições, com o intuito de reduzir o trabalho de criar várias categorias com o mesmo contexto. Podemos usar essas substituições para trocar palavras em formar de gírias por sentenças formais, desse jeito facilita o entendimento do bot de acordo com a sua base de conhecimento, trocando abreviações, por palavras como, por exemplo: substituir “cmg” por “comigo”.

As substituições estão presentes em um arquivo em XML, que contém todas as

possibilidades, ou grande parte delas, geralmente, previstas pelo *botmaster*[2] (define o componente de personalidade) para que o bot de conversação consiga entender o contexto da sentença. Na Figura 23, temos um trecho do arquivo que usa a *tag* de *substitute find* e *replace*, responsáveis por fazer as substituições.

```
<substitute find= "vc" replace=  "voce" >
<substitute find= "pq" replace= "porque">
<substitute find= "cmg" replace= "comigo">
<substitute find= "nois" replace= "nos">
<substitute find= "td" replace= "tudo">
<substitute find= "tdo" replace= "tudo">
<substitute find= "blz" replace= "beleza">
```

Figura 23 - Trecho de um arquivo em AIML contendo as *tags substitute* e *replace*.

g) Tag system

A *tag system*, permite que o interpretador execute comandos do *shell*, assim é possível executar através do interpretador comando de sistema que não são previstos e permitir interatividade com banco de dados, e outros sistemas, inclusive gerência de rede através do *chatterbot* (LEONHARDT, 2005).

Shell de comando é um software que oferece comunicação direta entre o usuário e o sistema operacional. A interface de usuário não gráfica do *shell* de comando é o ambiente propicio para a execução de aplicativos e utilitários baseados em caracteres. Este componente executa programas e exibe os dados de saída em uma tela usando caracteres individuais de forma idêntica ao interpretador dos comandos do *MS-DOS*.

Pode-se usar o *shell* de comando para criar e editar arquivos em lotes e, assim, automatizar tarefas rotineiras. É possível usar o *Windows Script Host*, *CScript.exe*, para executar scripts mais sofisticados no *shell* de comando. O uso de arquivos em lotes possibilita a realização de operações mais eficientes do que através da interface do usuário. Os arquivos em lotes aceitam todos os comandos disponíveis na linha de comando. Um exemplo de uso da tag system, pode ser visualizado na Figura 24. Neste exemplo, o sistema retorna ao usuário a hora do sistema.

[2] *Botmaster* é o programador do *chatterbot*, a pessoa responsável pela criação e manutenção da base de conhecimento de um robô de conversação (CORREA, 2001).

```
<category>
        <pattern> QUE HORAS SAO </pattern>
          <template> Sao <system>date</system></template>
</category>
```

Figura 24 - Exemplo de uso da *tag system*, em que o sistema informa a hora ao usuário executando comando do *shell* do sistema

3.4.3 Construção da base de conhecimento

Um dos principais problemas encontrados na maioria dos *Chatterbot* é a falta de uma base de conhecimento consistente que permita ao *Chatterbot* responder perguntas diferentes daquelas consideradas óbvias sobre um determinado assunto.

O grande desafio dos pesquisadores da área é identificar a forma como as pessoas elaboram perguntas sobre dúvidas de um determinado domínio do conhecimento na Internet. Algumas técnicas baseadas na psicologia rogeriana conseguem estabelecer um diálogo, entretanto, elas não são capazes de responder perguntas específicas que podem ser elaboradas de diversas formas (WEIZENBAUM, 1966).

Na busca por alternativas que agilizem a construção dessas bases de conhecimento formulou-se uma proposta constituída a partir das seguintes etapas:

1. Definir os padrões a serem recuperados de acordo com uma metodologia baseada na linguística de corpus. Através da observação dos resultados gerados com diversos tamanhos de padrões, verificou-se que um conjunto de 16 palavras forma uma boa estrutura para auxiliar na identificação do padrão de conversação adequado a especificidade desse trabalho;
2. Definir os objetivos e utilidades padrões;
3. Definir a categorização e o tratamento que será executado nos arquivos;
4. Definir o processo de manipulação que será feito pelos especialistas no domínio do conhecimento e qual o layout dos arquivos após esse processo;

3.4.4 Técnicas e Estratégias de Resposta

Segundo Cassell (2003), existem diferentes estratégias para a escolha de uma resposta mais apropriada a cada interação de um *chatterbot* com um usuário. As principais técnicas de seleção de respostas são:

Casamento de padrões - É uma técnica em que é feito um casamento entre um conjunto de palavras-chave e um grupo de sentenças referentes às palavras-chave. Em geral, é definida certa prioridade entre as respostas, dessa maneira, para uma pergunta que contenha uma determinada palavra "mãe", uma possível resposta poderia ser selecionada de um grupo de respostas relacionadas ao tema "família";

Rede de ativação de respostas - É uma técnica baseada em respostas passadas. Consiste em vários padrões com respostas associadas a um valor de ativação associado a cada nó contendo a resposta. Caso o usuário mencione algum padrão específico, a ativação é estimulada nos nós relacionados e inibida nos outros nós (MAULDIN, 1994);

Raciocínio baseado em casos - O programa possui um conjunto de casos passados na sua base permitindo percorrer o conjunto de dados à procura de uma solução e responder, adaptando ao caso atual.

Em se tratando de conversação e da forma como se deve iniciar, direcionar ou manter uma conversação com um usuário, muitos trabalhos já vêm sendo desenvolvidos traçando uma analogia com a forma humana de abordagem.

De acordo com Mauldin (1994), os *chatterbot* fazem uso de diversas estratégias para dar a ilusão de inteligência e fluência, incluindo:

- Manter a iniciativa através do constante questionamento;
- Incluir partes da pergunta do usuário na formação da resposta;
- Mudar o nível de conversação através de perguntas que aprofundem o diálogo, como, por exemplo: Porque você me perguntou isto?";
- Permanecer maior tempo possível no mesmo tópico, questionando o interlocutor quando o mesmo surpreendentemente mudar o assunto;
- Começar um novo tópico quando a conversa se tornar muito repetitiva;
- Fazer comentários controversos e humorísticos sobre algum assunto que seja foco da conversação.

3.4.5 Interpretadores

O desenvolvimento do interpretador para AIML foi iniciado por Richard Wallace. (WALLACE, 2007) A primeira versão de uso de interpretadores não chamou muita atenção dos programadores da época, e passaram a JAVA, que inicialmente foi chamado de *Program* A, e depois passou a ser chamado de *Program* D. Essa implantação de uma linguagem de programação, empenhou o uso de outras linguagens como interpretador, popularizando a utilização de AIML como linguagem padrão para construção de *chatterbot*.

Cada interpretador possui um conjunto de particularidades que permitem a execução de um bot no processo de conversação, além de possuir a capacidade de reconhecer os principais elementos de uma linguagem, como armazenar o diálogo em um arquivo de log, ou a última conversação temporariamente, simulando uma espécie de aprendizado.

Atualmente, dispomos de muitas opções disponíveis de interpretadores AIML, sendo que maior parte são *opensource* (ferramentas de uso livre) possibilitando acrescentar novos recursos. A seguir destacamos os interpretadores mais usados (WALLACE, 2007):

- *Program* D: é implementado em JAVA, podendo atuar na web, programas de mensagem instantânea, e com funções para registro de conversações;
- *Program* E: mais conhecido como *PHiliP*, é escrito na linguagem PHP com MySQL, armazena a base de conhecimento AIML do bot para o banco de dados, permite a integração de interfaces em Flash, HTML e XML-R.
- *Program* M: Interpretador implementado em uma linguagem conhecida como SETL (*Set Theory and Mathematical Logic*). Mais uma especificação formal para AIML do que um aplicativo de trabalho prático, *Program* M é uma das implementações mais compactas em pouco menos de 1000 linhas de código. Esse interpretador é utilizado somente em Linux, e na Tabela 1, são mostrados os arquivos de conFiguração e classe AIML, que são usados para construção e estruturação de um bot usando esse modelo de interpretador.

default.aiml	A categoria AIML padrão se nenhuma outra AIML existe.
files.txt	Uma lista de arquivos AIML para carregar.
macrosubs.txt	AIML substituições macro.
delims.txt	Delimeters Sentença divisão.
abbrevs.txt	Substituições abreviação.
formalsubs.txt	Substituições formais de fala.
normsubs.txt	Substituições de normalização sentença.
person2subs.txt	1 ª Pessoa - 3 swaps Pessoa pronome.
personsubs.txt	1 ª Pessoa - 2 swaps Pessoa pronome.
defaults.txt	Valores padrão predicado.
pronouns.txt	Pronome pessoal predicados.

Tabela 1 - Arquivos e classe com arquivos de conFiguração do *Program* M

• *Program* N: Criado por Gary Dubuque foi projetado principalmente, para ser usado como um "assistente pessoal". É um interpretador compatível AIML implementado em C++. Extensões são permitidas para um intérprete AIML, mas o intérprete deve ser capaz de analisar o código AIML, tal como definido por um padrão AIML.

• *Program* P: mais conhecido como PASCALice, tendo sido desenvolvido em Delphi e é liberado sob a licença GNU GPL. Também fornece uma ferramenta de verificação de AIML chamada *ShadowChecker*.

• *Program* R: é implementado em *Ruby on Rails*, entretanto precisa passar por uma série de testes, buscando obter uma otimização em seu desempenho.

• *Program* V: é uma implementação do Perl de um intérprete AIML. Depende do uso padrão da indústria de *mod_perl* (para Apache) para que o servidor web possa lidar com muitas das tarefas não relacionadas com AIML que outros intérpretes (como *Program* D) lidam.

• *Program* Y: Mais conhecido como PyAIML, é um interpretador AIML implementado como um pacote Python 100% puro. O pacote possui cerca de 1000 linhas de código, sem dependências externas além das bibliotecas padrão do Python. Permite a adição de recursos avançados encontrados na maioria dos outros pacotes.

• *Program* Z: É implementado em Common Lisp, e foi desenvolvido pela organização *PandoraBots*, sendo a forma mais rápida de desenvolver um robô de conversação, possuindo um serviço de hospedagem de *chatterbot* e ferramentas para o desenvolvimento destes, permitindo criar e publicar os *chatterbot* através de qualquer navegador web.

• *Program* #: intérprete em AIML.net, sob a GNU GPL. Foi desenvolvido usando a versão 1.1 do *framework.net*, usando integração em aplicações *Windows.Forms*, serviços do Windows e Web Services.

No desenvolvimento da aplicação em pauta descrita nesse trabalho, o interpretador utilizado para inferência na base de dados AIML é *Program* Y, também chamado PyAIML que utiliza a linguagem de programação Python. Na sessão a seguir será descrito características da linguagem que justificaram o seu uso no desenvolvimento da aplicação.

3.4.6 Linguagem de Programação Python

De acordo com Feitosa (2008), Python possui as seguintes características: Curva de Aprendizado: Sua sintaxe é simples, facilitando a aprendizagem da linguagem Python.

1. Conciso: Código escrito em linguagens com tipagem dinâmica, como Python, tende a ser menor do que código escrito na maioria das outras linguagens.
2. Fácil de Ler: O Python é por vezes referido como "pseudocódigo executável". Enquanto isto é claramente um exagero, programadores mesmo sem muito conhecimento de Python código podem ler e entender facilmente o que o código realiza.
3. Coleta de Lixo Automática: Trata-se de um sistema que elimina os erros causados pelo acúmulo de dados inúteis na memória do computador.
4. Facilmente Extensível: Python padrão vem com várias bibliotecas, inclusive aquelas para as funções matemáticas, análise de XML, e download de páginas da web.
5. Interativo: Python pode executar programas diretamente a partir da linha de comando, e ele também tem um *prompt* interativo que permite que você escreva funções, chamadas, criar objetos, e testar pacotes interativamente.
6. Multiparadigma: Python suporta os paradigmas de orientação a objetos, procedural, e funcional. Algoritmos de aprendizagem de máquina variam muito, e a maneira mais clara de se implementar um pode usar um paradigma diferente de outra implementação.
7. Multiplataforma e livre: Python tem uma única referência à execução de todas as principais plataformas e é gratuito para todos eles. A aplicação desenvolvida nesse trabalho é compatível com os sistemas operacionais Windows e Linux.

Python já vem instalada na maioria das distribuições Linux, e com o interpretador já executando diversos outros programas, dessa maneira é uma opção menos intrusiva, não sobrecarregando o computador do usuário com mais pilha de programas, e de recursos

computacionais tais como memória e CPU necessários para poder utilizar o assistente, e esta característica pode facilitar a adoção do assistente pelos mantedores das distribuições de GNU/Linux, para que ele já venha instalado por padrão posteriormente.

4 ESTADO DA ARTE

Nesta seção, será feita uma breve abordagem sobre a estrutura do *chatterbot* tido como base para o desenvolvimento e construção da base de conhecimento do Systembot. O nome desse *bot* é *Equus*, cujo significado em latim quer dizer “cavalo”, e é voltado para o ramo da equinocultura, com o objetivo de disseminar o conhecimento da cultura, da criação e manipulação de equinos (FILHO, 2009).

A utilização de conceitos de IA no processo de construção dessa aplicação visa auxiliar e proporcionar conhecimento relacionado a conceitos sobre equinocultura, através de um de interação humano-computador.

Equus utiliza técnicas de construção, que foram apresentadas anteriormente, e o intuito de seu desenvolvimento foi a criação de um produto que difunda conhecimento a população e seja de fácil construção viabilizando o seu uso em outras áreas.

Chatterbot podem atender a várias situações que vão desde atendimento a um cliente, como a uma importante fonte de pesquisa, proporcionando interatividade e estimulando a propagação do conhecimento. A construção de mecanismos para otimizar sua eficiência tem ganhado expressão e pesquisas.

O objetivo da construção de Equus, foi garantir uma fácil implementação de um Bot, utilizando apenas ferramentas livres, além de soluções, para que seja possível expandir e aprimorar o projeto.

É disponibilizado um interpretador de AIML, uma base de conhecimento genérica, responsável pelo armazenamento de informações de conhecimento geral e uma base de conhecimento específica voltada para assuntos relacionados ao tema equinocultura.

4.1 Conhecimento Específico

A construção de um *chatterbot* com um conhecimento genérico leva uma grande parcela de tempo, principalmente por ser necessário alimentar e agregar conhecimento à base de conhecimento, de maneira supervisionada pelo *botmaster*.

Apesar de ter grande quantidade de informação, o bot pode não conseguir abordar todo o contexto relacionado a outros assuntos relacionados ao ramo da Equinocultura,

entretanto, a elaboração de um sistema com um conhecimento especifico ajuda a reduzir os padrões nos documentos AIML, além de se tornar uma importante fonte de consulta de informações sobre um determinado assunto.

Filho (2009) constatou a inexistência de um modelo de *chatterbot* operante na equinocultura, que representa atualmente uma importante faixa do agronegócio brasileiro. Isso justificou a decisão de criação de um robô de conversação, considerando que o maior benefício desta abordagem é o uso de técnicas de IA na aplicação de um *chatterbot* em um contexto ainda pouco explorado.

4.2 O Interpretador

O interpretador utilizado na construção do *chatterbot* Equus, foi o *Program E*, conhecido como "PHiliP ", que trata-se de uma implementação na linguagem PHP disponível pela comunidade *alicebot.org*.

Filho (2009) usou a plataforma em PHP, pelo fato da mesma possuir vantagens quanto à portabilidade, facilidade de hospedagem do serviço e crescimento da comunidade e utilizadores. O seu uso possibilita, a utilização de um projeto de sistema facilmente escalável e portátil, buscando obter maior visibilidade, sendo voltado para Web.

4.3 Base de Conhecimento

A base de conhecimento de Equus possui todas as informações necessárias, para que seja possível realizar o processar informações e realizar um diálogo do bot com os usuários. A construção dessa base seguiu duas etapas, uma utilizando a base de conhecimento padrão, gerada para atender um contexto comum de entradas com mais frequência de utilização, e a outra etapa, foi a utilização da ferramenta construída para gerar conhecimento AIML.

O *chatterbot* Equus é uma iniciativa para disseminar uma área de conhecimento, que atua com poucos padrões de conhecimento, sendo possível contar com a sua ajuda para auxiliar e popularizar informações relacionadas a equinocultura. Possui uma arquitetura simples, que permite sua funcionalidade, e o uso de sua ideia para a construção de outros Bots

com conhecimento em outras áreas, e outros propósitos, trafegando e disponibilizando informações por toda Web.

4.4 Modelo Funcional

O *chatterbot* Equus é divido em 4 funções principais, que em conjunto denotam o funcionamento do *Chatterbot*. Na Figura 25, temos uma representação do modelo funcional desse sistema.

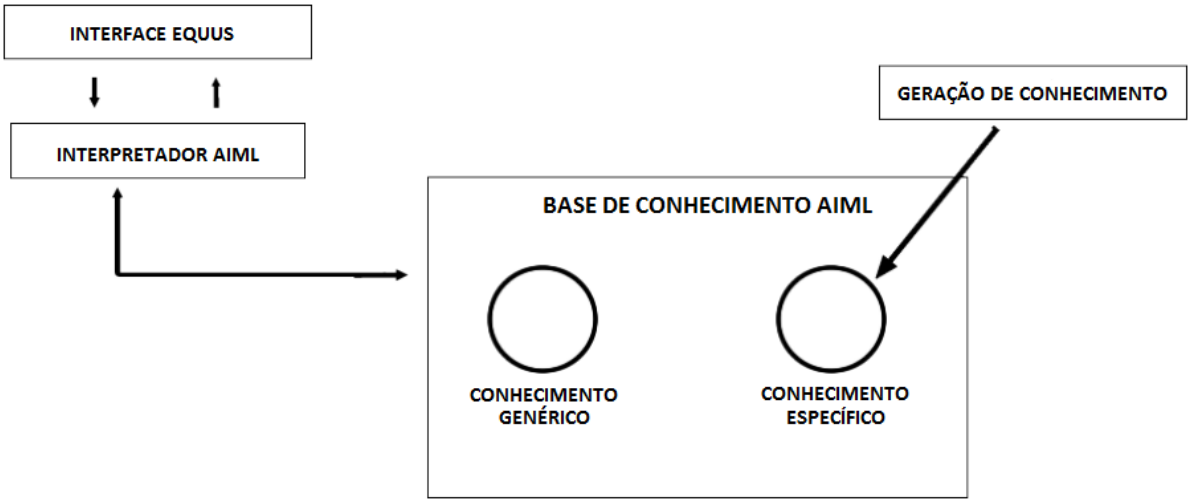

Figura 25 - Modelo Funcional de Equus
Fonte: (FILHO, 2009).

O componente interface é o responsável por receber as mensagens recebidas do usuário e mostrar a resposta do bot, de acordo com a sentença de entrada.
Essa sentença de entrada, fornecida pelo usuário, é utilizada pelo interpretador para buscar na base de conhecimento AIML uma resposta convincente.

A outra base de conhecimento foi gerada através de uma ferramenta de geração de conteúdo AIML, que consegue abstrair de um contexto específico, para a base AIML. Equus foi modelado para que uma implementação para outra área de atuação seja prática, e torne seu uso facilitado utilizando todas ferramentas livres.

4.5 Geração de Conteúdo Específico

A ferramenta construída utiliza um mecanismo facilitador para a construção e manipulação das bases AIML, agilizando o processo de implementação do conhecimento para o *chatterbot*. Para o funcionamento da ferramenta geradora de conhecimento, utiliza-se um dicionário de termos técnicos relacionados com o tema a ser abordado no diálogo.

A Figura 26 mostra uma representação do propósito de uso da ferramenta de geração de conteúdo AIML.

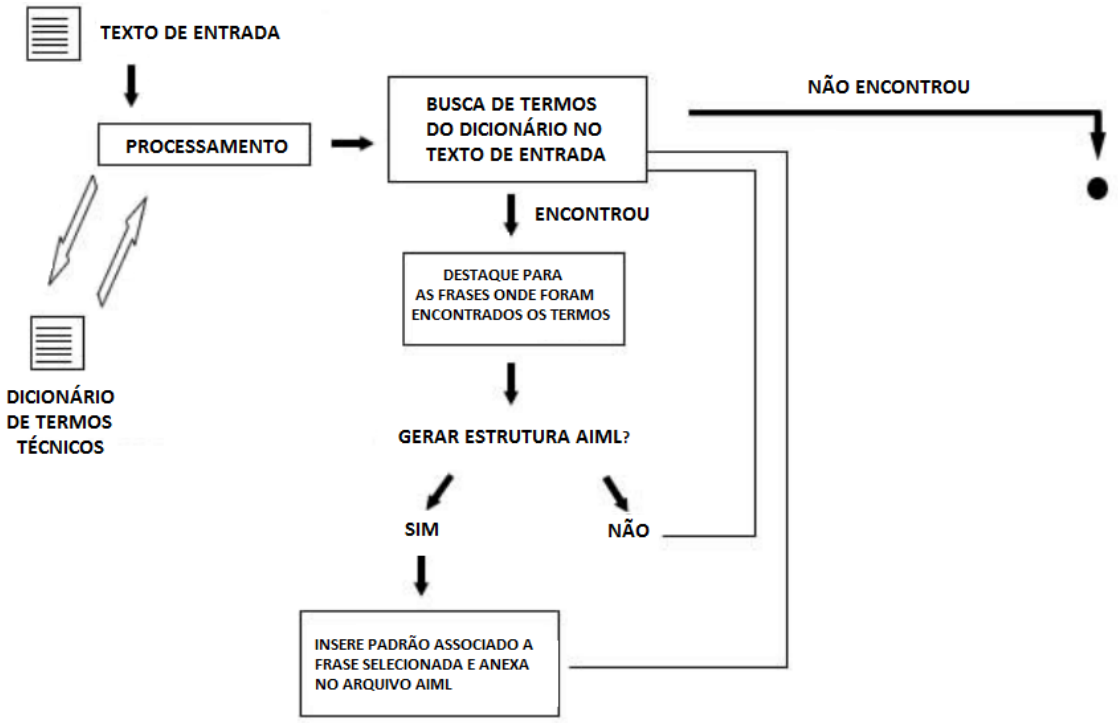

Figura 26 - Funcionamento da ferramenta de geração de conteúdo AIML
Fonte: (FILHO, 2009).

Os elementos do processo de criação do conteúdo em AIML são: o texto de entrada, o dicionário de dados e a criação das categorias AIML. Na entrada, utiliza-se o texto de entrada fornecido pelo usuário, e em seguida, acontece o processamento, que consiste em buscar as palavras-chave no dicionário de termos técnicos. O processo de geração de conhecimento é simples, e funciona como um módulo independente, para auxiliar na inserção de conhecimento na base, pelo *Botmaster*. Para a construção e inserção do conhecimento, é criado um documento texto, que contém as palavras-chave, e (ou) termos técnicos referentes a

uma área específica de conhecimento, que no caso é a equinocultura.

5 O *Chatterbot* SYSTEMBOT

Chatterbot podem ser utilizados como uma ferramenta de interação humano-computador, além de também permitirem auxiliar no processo de aprendizado buscando promover o estimulo dos envolvidos, sendo capazes de explorar o comportamento do usuário, por meio de padrões, podendo até mesmo influenciá-lo em um diálogo. Na seção anterior, foi mencionado que a linguagem de marcação AIML, é a linguagem mais difundida no processo de construção e desenvolvimento da base de conhecimento dessas aplicações.

A motivação do desenvolvimento de um *chatterbot*, é proporcionar de maneira dinâmica, o acesso de informações relacionadas ao curso de Bacharelado em Sistemas de Informação – UFPI (CSHNB), aos alunos ingressados no curso, promovendo interatividade e propagação de conhecimento através de um ambiente virtual.

Esse projeto teve como intuito, propor a construção de um *chatterbot* de fácil implementação, que utilize ferramentas *opensource*, sendo possível permitir a expansão e aprimoramento desse trabalho por meio das técnicas utilizadas para a sua construção.

Neste capítulo, será colocado em prática o que foi citado nos capítulos anteriores. Entre os tópicos a serem abordados no decorrer do capitulo, serão abordados, a proposta do trabalho, sua arquitetura, o processo de conversão dos códigos em Python através do Pymc, a interface e o funcionamento da aplicação, e os resultados obtidos pela aplicação, cujo nome denominei Systembot, que se enquadra na categoria dos *chatterbot* denominados Faqbots.

5.1 O Trabalho Proposto

O desenvolvimento desse projeto, teve como principal intuito, implementar um ambiente que proporcionasse a interação com os alunos do curso de Bacharelado em Sistemas de Informação – (CSHNB), através de uma aplicação, cujo nome foi denominado Systembot,

que busca fornecer informações relacionadas ao curso de Sistemas de Informação, sobre os funcionários que trabalham nos setores, a estrutura, disciplinas e professores ministrantes.

São disponibilizados, um interpretador AIML (*PyAIML*), as bases de conhecimento AIML da aplicação, que são responsáveis por armazenar as informações de conhecimento genérico do bot e específico, e o projeto gráfico desenvolvido em linguagem C, que após a utilização do *Pymc*, teve sua linguagem convertida para a linguagem *Python*. Essa aplicação será disponível inicialmente para sistemas de plataforma *Windows*.

O conhecimento genérico do bot, refere-se ao comportamento padrão da aplicação em um diálogo. O conhecimento especifico, se refere ao comportamento do *chatterbot*, quando o assunto em questão são assuntos relacionados ao curso de Sistemas de Informação.

5.2 A Interface e Conversão em códigos Python (Pymc)

Um *chatterbot* possui um ambiente virtual que podemos denominar Interface usuário. Neste modulo, criamos uma parte gráfica que receberá a sentença de entrada fornecida pelo usuário e a sentença irá ser analisada pelo interpretador que utilizará a base de conhecimento AIML, fazer uma consulta na base e retornar a resposta. Chamamos esse processo de *Feedback*, que baseia-se no modelo funcional da estrutura de um *chatterbot*. Na Figura 27, é mostrada a interface da aplicação desenvolvida através da ferramenta *Qt Creator*.

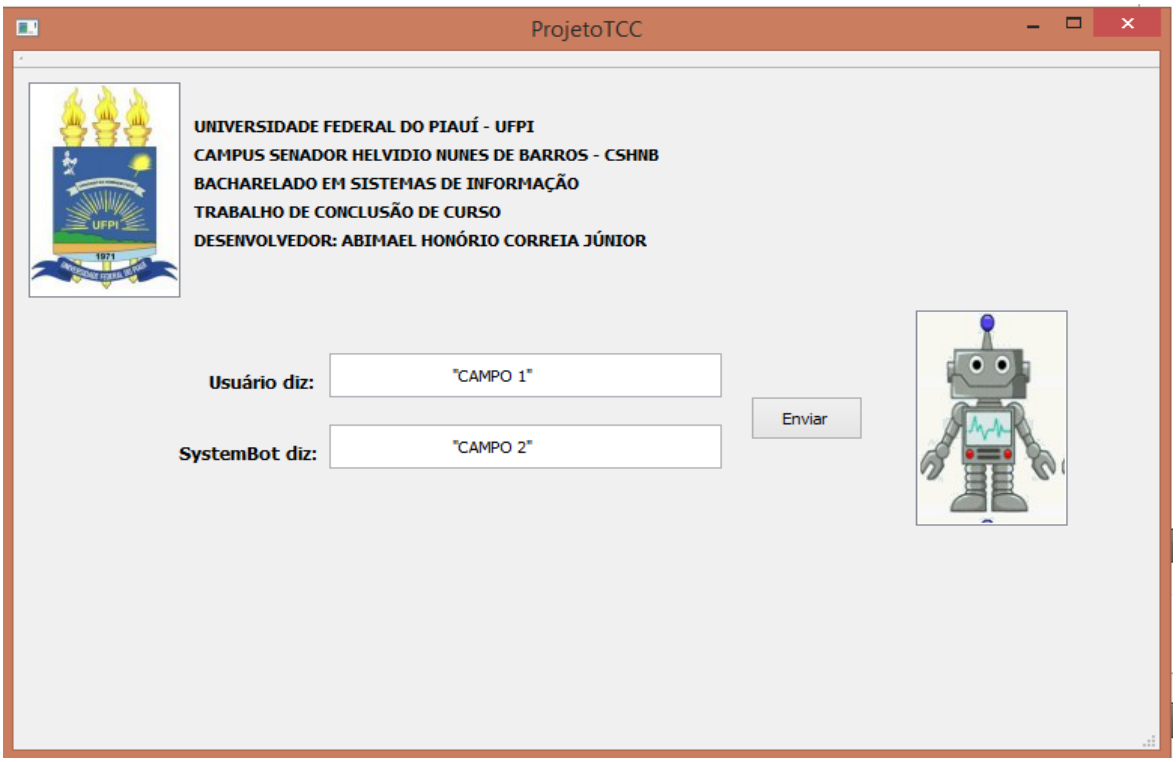

Figura 27 - Modelo de Interface do Systembot

O desenvolvimento da interface foi baseado nos modelos propostos pelo Projeto PandoraBots[3], que mostra o registro do log ao usuário abaixo do campo de entrada de dados. Essa interface é composta por dois campos, o primeiro campo denominado "campo 1" é o responsável pela entrada de dados, ou seja a sentença enviada pelo usuário. O "campo 2", é responsável por mostrar ao usuário, qual a resposta, de uma determinada sentença, do bot ao usuário.

Para realizar essa conversão, é utilizado o comando Pyuic4, que converte o arquivo padrão da ferramenta, em códigos nativos da linguagem Python, permitindo que seja possível o interpretador ler e executar a aplicação, juntamente com a interface criada. Para realizar essa conversão, usamos o seguinte código descrito na Figura 28.

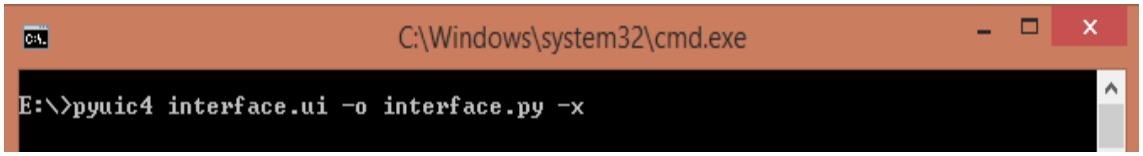

Figura 28 - Conversão do arquivo padrão ".ui" para códigos Python

[3] O Projeto PandoraBots, é um projeto de desenvolvimento de robôs virtuais que podem ser disponibilizados na internet para outros usuários.

Nessa Figura, nota-se que o arquivo gerado pela interface de formato *.ui* é passado como parâmetro de entrada e será convertido em um novo formato *.py* de saída. A opção "–x", presente no final da linha de comado é responsável por gerar um arquivo executável, após o final da execução.

5.3 Arquitetura do Systembot

A arquitetura adotada na construção do *chatterbot* foi dividida em um modulo principal (módulo de conversação), interface com o usuário e o elemento de execução, todos interligados entre si e gerenciados pelo interpretador.

Descrevendo os componentes dessa arquitetura temos:

- Módulo de conversação: Camada formada pelas bases de conhecimento que são montadas de forma a funcionar como FAQs. O Systembot utiliza um mecanismo que utiliza log de interações com usuários para simular o aprendizado, esse mecanismo permite que a aplicação utilize recursos do diálogo para interagir com o usuário, simulando, ter entendido e observado tudo que o usuário digitou durante a conversação;
- Modulo Interface com usuário: Constituída de uma tela gráfica, criada através da ferramenta Qt Creator (versão 5). Essa interface é o ambiente virtual do bot, que permite a interação do usuário, através da inserção de sentenças de entrada emitidas pelo usuário, e as respostas do *chatterbot*. Esse componente pode ser utilizado ou não pelo Systembot, visto que, também é possível interagir com o usuário em modo texto. Nas Figuras 28 e 29 temos exemplos desta aplicação funcionando nos dois modos (gráfico e texto).

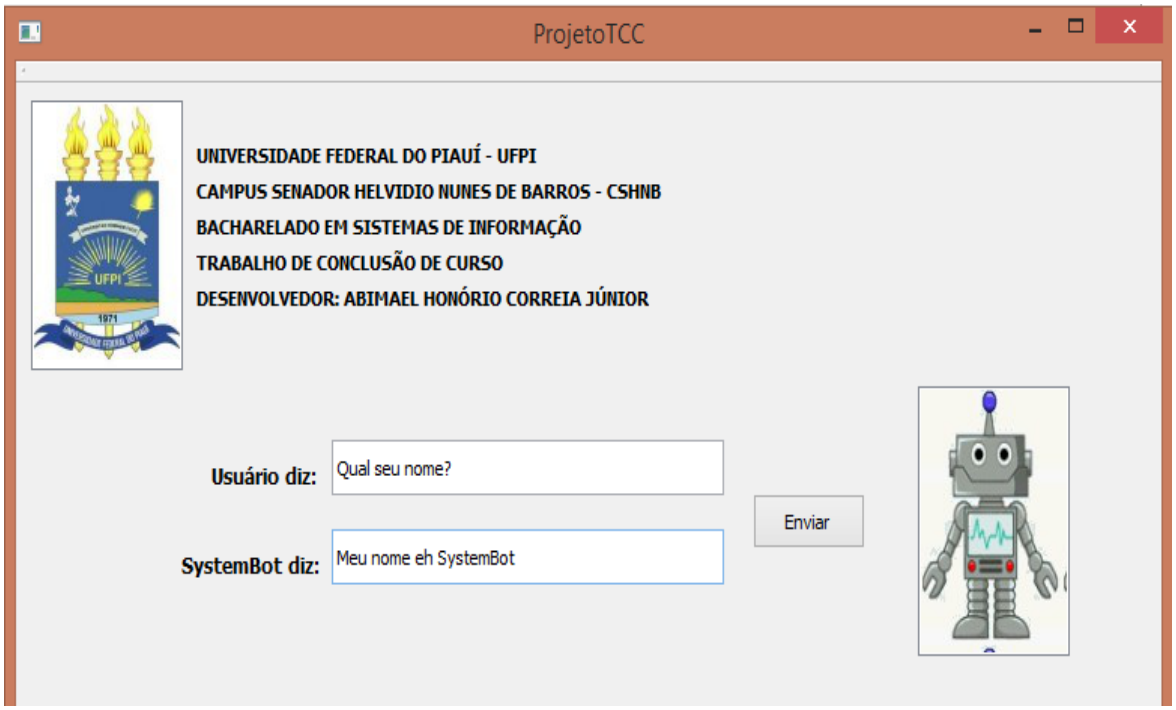

Figura 29 - Systembot em Modo Gráfico

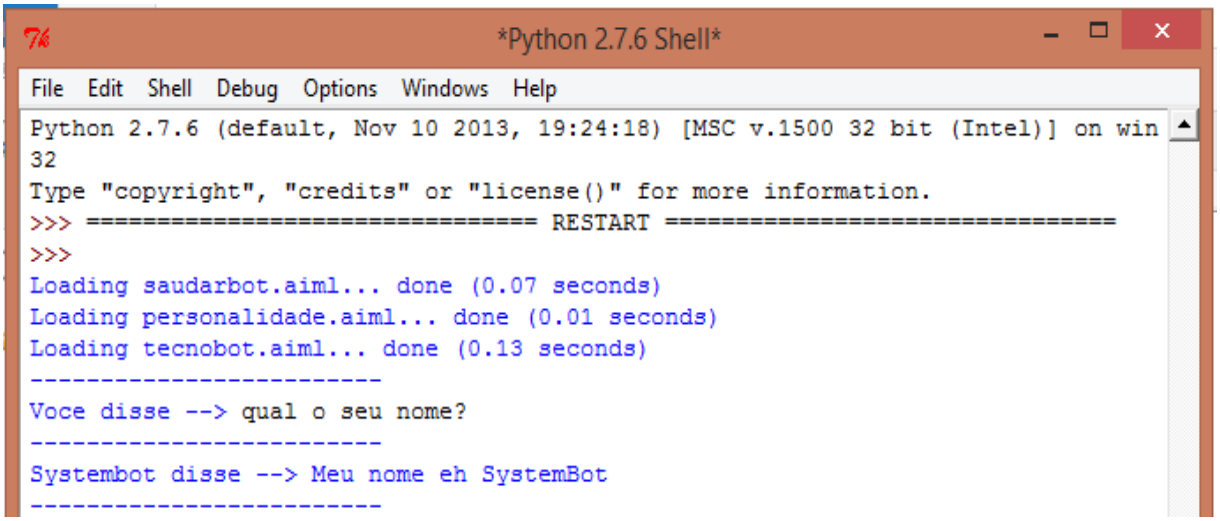

Figura 30 - Systembot em Modo Texto

• Camada de Controle: Parte central da aplicação. A função desse componente é analisar as sentenças de entrada inseridas pelo usuário, no componente “interface”, e fazer a consulta no modulo de conversação que é composto pelas categorias da base de conhecimento AIML, obtendo as respostas, e aplicando a estratégia de seleção da melhor resposta às sentenças dos

usuários.

O módulo de conversação é responsável por receber a entrada do usuário, em linguagem natural, interpretá-la, e achar a melhor resposta para o usuário na base de conhecimento AIML, que é usada para consulta aos padrões de entrada e saída e análise de substituições de abreviaturas ou palavras informais.

Observando sua arquitetura, na Figura 31, vemos que o Systembot é modular, apresentando 2 módulos interligados (interface e conversação), e independente de domínio de aplicação, pois permite que componentes possam ser utilizados separadamente em uma outra aplicação. Como exemplo disso, temos a base de conhecimento que pode ser usada em uma aplicação Web, hospedada no Projeto PandoraBots a fim de favorecer sua disponibilidade e usabilidade de seus componentes (bases de conhecimento).

Com base nessa Figura, no módulo interface, o usuário realiza as perguntas através do componente interface. A camada de controle, que é o componente presente nos dois módulos (interface e conversação), é responsável por receber as informações que provêm da interface e converte-as na linguagem de máquina. Em seguida, as sentenças convertidas pela camada de controle são gerenciadas pelo interpretador, que tem a função de analisar as sentenças, e fazer uma busca nas bases de conhecimento. No módulo de conversação, são realizadas as conversões das sentenças e buscas na base de conhecimento. Ao realizar uma busca nas bases, é retornado para a camada de controle a resposta referente à pergunta em questão, e após esse processo, através do componente interface, o usuário visualiza a resposta final.

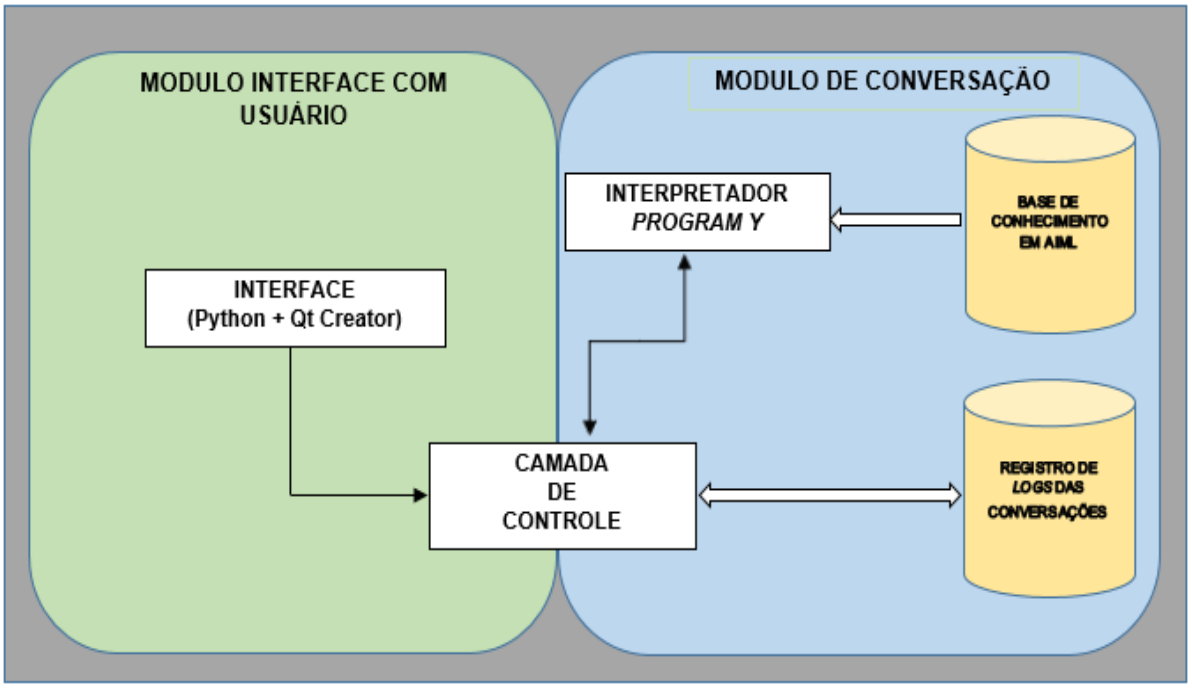

Figura 31 - Arquitetura de Systembot

Um aspecto interessante nessa aplicação, é que quando não consegue responder alguma sentença de entrada do usuário, deixa claro que não entendeu o contexto, solicita que o usuário refaça uma nova pergunta ou tenta iniciar um novo tema com o assunto em pauta, conforme exemplo na Figura 32. Dessa maneira, sempre que o *chatterbot* não entender uma sentença emitida pelo usuário, o padrão de resposta da aplicação emite mensagens solicitando que o usuário refaça a pergunta, ou tente mudar de assunto, permitindo que o diálogo em questão não seja encerrado.

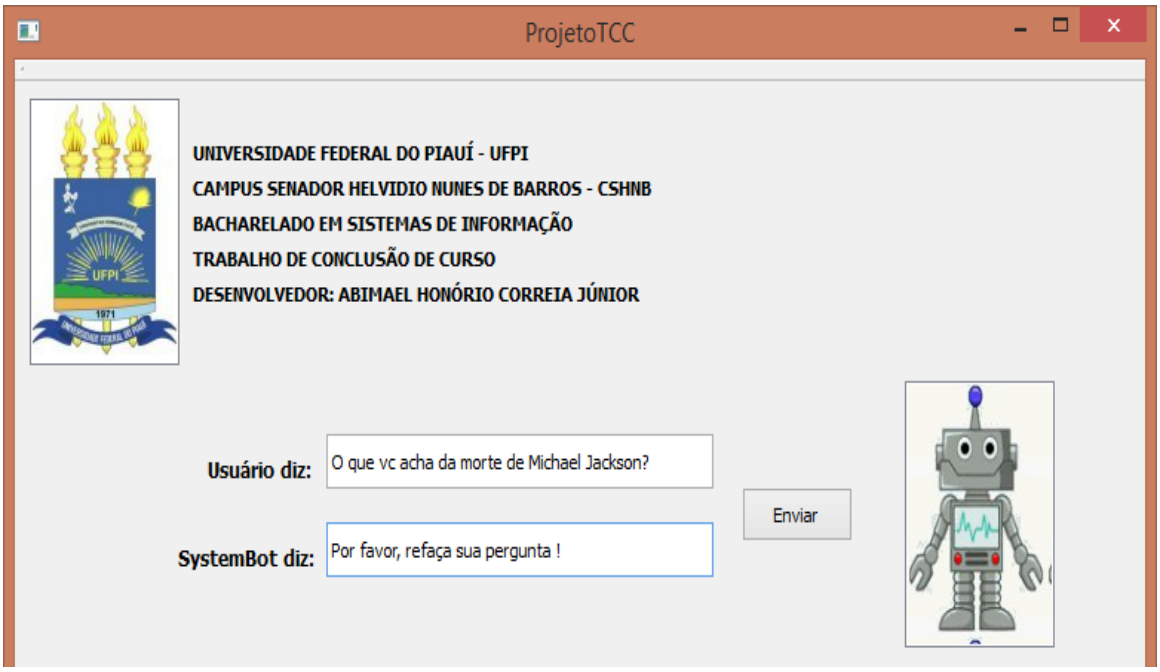

Figura 32 - Systembot solicita que o usuário refaça a pergunta.

5.4 Etapas de Construção da Base de Conhecimento

Na construção da base de conhecimento de um *chatterbot*, é necessário que exista a capacidade do bot de agregar conhecimento, mesmo que parte do conhecimento da aplicação seja inserida de maneira manual pelo *botmaster*.

Não pode-se minimizar o fato de que, pode acontecer a possibilidade do *chatterbot*

não responder com clareza uma determinada sentença, mesmo que a base possua muitas categorias. É necessário criar mecanismos que possibilitem que o bot continue interagindo com o usuário, mesmo que não entenda o diálogo, ou parte do diálogo em questão.

Dessa maneira, o bot pode vir a abordar de maneira clara todo o conteúdo existente de assuntos mencionados em um diálogo, assim, a presença de um conhecimento especifico permite a redução de criação de padrões nos arquivos aiml sobre um tema em foco. Para a construção dessa base de conhecimento, foi adotada a metodologia utilizada por Filho (2009), que define quatro etapas fundamentais no desenvolvimento dessa aplicação.

Na Etapa inicial, que foi denominada Etapa 1, foi realizado o início da construção da base de conhecimento, com a criação de categorias responsáveis por identificar os padrões de uma sentença emitida pelo usuário, e retornar uma mensagem, de acordo com essa sentença.

Na próxima etapa (Etapa 2), foi definido o padrão de comportamento do *chatterbot*, e sua personalidade. Essa definição indica como a aplicação reage a uma sentença do usuário, quando o mesmo inicia o diálogo, insulta o bot ou quando se despede.

Após realizar essas duas etapas, foi iniciado o processo de testes e analises do bot. Nessa etapa (Etapa 3), o comportamento do bot foi testado, onde visa-se perceber aspectos como precisão da resposta, prosseguimento do diálogo, desempenho. Esses aspectos, são muito importantes, pois, permitem indicar se o comportamento do *chatterbot* é suficiente para caracterização da simulação de um ser humano durante um diálogo.

Na Figura 33, é apresentado um trecho da base de conhecimento de Systembot. Neste trecho, temos a palavra “COORDENADOR *” presente em uma sentença de entrada, e dessa maneira, quando uma sentença tiver a presença dessa palavra, o bot poderá retornar informações como o nome do coordenador do curso, e informações sobre disciplinas que ele também ministra no curso.

```
<aiml version=1.0>
<category>
<pattern> COORDENADOR * </pattern>
        <template> A coordenação atual do Curso de Sistemas de Informação tem como coordenador a
        Msc. Patricia Medyna </ template>
</category>
<category>
<pattern> Patricia Medyna * </pattern>
        <template> Professora do Campus Senador Helvidio Nuns de Barros, é atualmente coordenadora
         do curso de Sistemas de Informação,
        e ministra aulas de disciplinas como Logica Computacional, e Programação Lógica.
        </template>
</category>
</aiml>
```

Figura 33 - Trecho da base de conhecimento AIML de Systembot.

Mesmo que possua uma grande quantidade de informação, o bot pode não abordar de maneira clara todo o conteúdo existente de assuntos mencionados em um diálogo, assim, a presença de um conhecimento especifico e uso de palavras-chave, permite a redução de criação de padrões nos arquivos *.aiml* sobre um tema em questão. Com base no trecho em AIML, com a utilização destes componentes pode-se obter respostas mais satisfatórias, pelo fato de referenciar ou atribuir uma palavra a um contexto, e pode ajudar a evitar povoar a base desnecessariamente, com a inserção de muitas categorias referentes a um único assunto.

5.5 SYSTEMBOT

O *Chatterbot* desenvolvido foi denominado Systembot, onde "*System*" em inglês, significa "Sistema", fazendo referência ao nome do curso que irá fornecer informações. Como foi mencionado anteriormente, o Systembot irá se comportar como um FaqBot, que responderá perguntas relacionadas a um determinado ponto em questão, nesse caso a estrutura do curso, os professores, coordenadores e disciplinas cursadas.

A utilização de conceitos da Inteligência Artificial na construção dessa aplicação, visa implementar mecanismos na construção de *chatterbot*, que possibilitem desenvolver um processo de auxílio ao usuário, respondendo e tirando dúvidas referentes a um assunto em

questão. Essas respostas, são baseadas em padrões palavra-chave, que foram mencionadas no capítulo 2.

Através de sentenças digitadas pelos usuários na interface do sistema, ele faz uma busca na base de conhecimento AIML, escrevendo respostas na mesma interface, utilizando também a memória do diálogo atual para fazer a escolha da melhor resposta a apresentar ao usuário.

Com esse comportamento, esse *chatterbot*, tem intuito de auxiliar alunos do curso de sistemas a terem acesso a informações do campus, essas informações estarão armazenadas em uma base de conhecimento armazenada em arquivos *.aiml*. Dessa maneira, os discentes poderão ter em suas máquinas, uma aplicação com as principais informações do curso, podendo serem esclarecidas através de um diálogo.

No seu funcionamento, a informação fornecida ao usuário é analisada pelo componente de processamento (camada de controle), e enviada a informação para a base de conhecimento AIML. Uma busca é realizada nas categorias da base, e baseado no melhor padrão de resposta para sentença em questão, a camada de controle recebe a informação e repassa para o componente interface, que é o responsável por mostrar ao usuário o resultado final do processo de interação entre usuário e máquina. Na Figura 34, temos a ilustração de como acontece esse processo de interação.

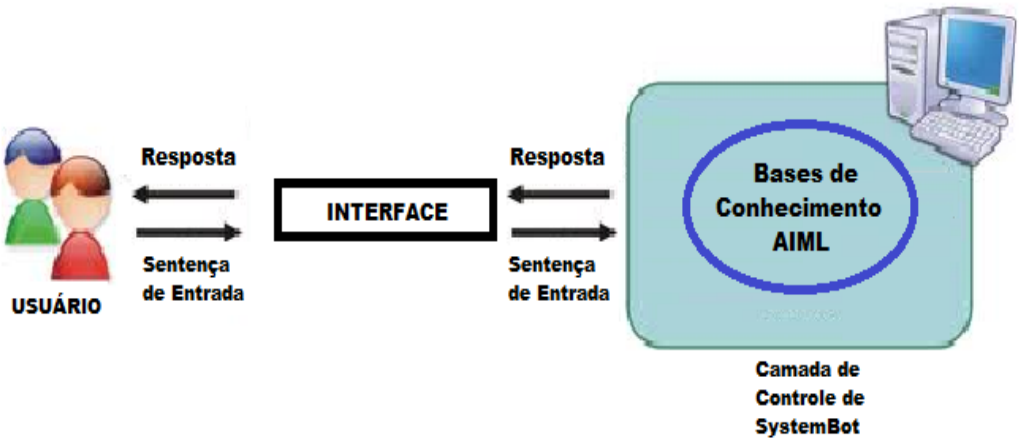

Figura 34 - Processo de interação entre usuário e o Systembot

Observando a Figura 34, mediante entrada de uma sentença, a aplicação irá retornar a informação, após a informação ser processada e ser analisada na camada de controle. Esse processo, de análise e resposta foi obtido através do uso da técnica de busca por palavras-chaves, para associação de dados.

O Systembot é uma ferramenta que pode ser muito útil para a disseminação de informações sobre os principais aspectos referentes ao curso de Sistemas de Informação, proporcionando uma forma comunicativa de repassar essas informações aos alunos do campus. A sua base de conhecimento, pode ser trabalhada e alimentada com o conhecimento sobre diversas áreas, podendo ter sua base alimentada e a partir daí poder trabalhar os padrões de entrada realizando o Tratamento de Sentenças, e mesmo com poucos padrões, pode-se chegar a um nível bem avançado com o decorrer do tempo, a partir de treinamentos e adição de conhecimento à sua base.

5.6 Resultados e Testes

Foram realizados experimentos com 27 (vinte e sete) alunos do curso de Sistemas de informações, onde estes foram designados a utilizar o *chatterbot* Systembot, para realizarem uma conversação com esse protótipo, e após essa etapa, avaliarem o uso, baseando-se na coerência e qualidade da interface do bot.

Os quesitos aplicados faziam referência respectivamente a (1) Qualidade do diálogo, (2) Interface e (3) Coerência do diálogo. Para obter uma estatística, foram atribuídos os valores das respectivas opções, e obteve-se uma média, que permitiu avaliar a qualidade da aplicação. Na Tabela 2, temos o resultado das avaliações feitas a Systembot.

Na Tabela 2, temos os campos com a expressão “O1, O2 e O3”, que fazem referência a quantidade de vezes que uma opção foi assinalada. Vale ressaltar, que essas 3 opções são divididas em: **1 – RUIM, 2 – REGULAR, 3 – BOM**.

Tabela 1 - Média das Avaliações

Quesitos	O1	O2	O3	MÉDIA
Qualidade do Diálogo	8	10	9	2,03
Interface	6	11	10	2,14
Coerência	5	10	12	2,25

Com base nessa análise e de acordo com a média geral obtida pelos alunos, observa-se que Systembot poderá atender as especificações e características mencionadas ao longo deste trabalho, desde que melhore significativamente o de qualidade do diálogo. Isso justifica-se pelo fato da quantidade de alunos que avaliaram a qualidade de maneira ruim, e é necessário melhorar muitos aspectos, que serão apontados em trabalhos posteriores. Quanto a interface, obteve-se a melhor média dentre os quesitos avaliados, isso indica que essa parte da aplicação teve êxito e melhor aceitação dos usuários.

5 CONCLUSÃO

A utilização de uso de *chatterbot* vem crescendo pelo fato de ter como principal atrativo o uso de uma linguagem natural, além de sua facilidade de interação com o usuário, sendo que o uso da solução representa um enorme potencial no sentido estimular o interesse e a curiosidade dos usuários.

A Inteligência Artificial tem buscado desenvolver diferentes meios de fazer com que um computador possa realizar suas tarefas de forma racional, não mais se prendendo a um conjunto pré-programado de instruções, e deste modo, envolvendo maior interatividade.

Neste trabalho foi abordado o processo de criação e desenvolvimento de um *chatterbot*, citando a sua evolução histórica, principais características e aplicações de uso até a construção do Systembot. Nesse contexto, foram realizados estudos buscando obter conhecimento sobre a linguagem AIML, que atualmente é a linguagem padrão usada para construção e alimentação da base de conhecimento de grande parte dos *chatterbot*. Outro aspecto em questão foi a maneira pelo qual AIML vem sendo utilizada, para aprimorar técnicas de seleção e estratégias de conversação de *chatterbot*, o uso das principais *tags* baseando-se na análise da conversação para o tratamento de intenções, permitindo realizar um diálogo coerente entre o usuário e máquina.

O objetivo desse trabalho foi desenvolver um *chatterbot* que possa interagir com os alunos do Curso de Sistemas de Informação (CSHNB), de maneira que seja estabelecido um processo de conversação em um ambiente virtual, sendo que a base de conhecimento pode ser alimentada manualmente, e após treinar o bot, permitir que seja adquirido conhecimento através de diálogos, estimulando esses alunos a adquirir informações de maneira interativa e dinâmica.

Para alcançar um resultado adequado, é necessário fazer um grande detalhamento das sentenças, a partir da entrada dos dados e melhorar de mapeamento das respostas. Pode-se verificar a questão do aprendizado do *chatterbot*, de tal maneira que o mesmo precisa passar por um processo de reavaliação da base de dados de conversação para que este aprendizado possa ocorrer, não sendo um processo autônomo. Os esforços não param e dentre os pontos mais interessantes para se estudar e solucionar esse problema, é interessante desenvolver mecanismos que proporcionem:

- A criação automática de conhecimento;
- Obter uma estrutura que analise o conhecimento adquirido pelo bot, observando os desvios

de linguagem, mudança de contexto repentina no diálogo;

• Buscar evolução no processamento de análise da língua portuguesa, vindo a desenvolver mecanismos que permitam manipular o conhecimento adquirido;

• Implementação de um sistema de busca, backup e recuperação mais efetivo de informações, documentos baseado em padrões textuais.

5.1 Trabalhos Futuros

Devido ter encontrado muitas dificuldades na implementação da aplicação, existem muitos aspectos que devem ser levados em consideração para que seja possível ter um sistema mais eficaz e que possa garantir resultados mais satisfatórios para os usuários. Os principais aspectos são:

• Desenvolvimento de um mecanismo de geração e tratamento de conhecimento autônomo;

• Implementação do o recurso de voz (*pyTTS*), permitindo que o bot possa interagir com os usuários tanto em modo visual, quanto em modo gráfico e com auxílio da voz.

• Aplicação de conceitos de inteligência artificial mais sofisticados e complexos, que possibilitem melhorar o desempenho, qualidade e maior coerência das respostas de uma sentença emitida pelo Systembot.

• Desenvolver um mecanismo que permita que o *chatterbot* possa enviar links que possam redirecionar o usuário para uma página web que possuam possíveis informações sobre um tema em questão no diálogo;

• Utilização de um *framework* para que Systembot possa ser utilizado na plataforma Web.

REFERÊNCIAS

ALICE. **AIML Reference Manual**. ALICEBOT, Rio de Janeiro, n.3, p.http://www.alicebot.org/documentation/aiml-primer.html, 2001.

AIRES, J. C. **Desenvolvimento de um *Chatterbot* para o Portal de Ensino da Universidade do Oeste de Santa Catarina**. 2008. 90f. Monografia (Bacharelado em Inteligência Artificial) - Universidade do Oeste de Santa Catarina, São Miguel do Oeste, SC, 2008.

Andrade, Adja; Jaques, Patrícia; Vicari, Rosa; Bordini, Rafael; Jung, João. **Uma Proposta de Modelo Computacional de Aprendizagem à Distância Baseada na Concepção Sócio-Interacionista de Vygotsky.** In: Workshop de Ambientes de Aprendizagem Baseados em Agentes; Simpósio Brasileiro de Informática na Educaçâo, SBIE 2000, 11., 2000, Maceió, Brazil. Anais... Maceió: UFAL, (2002).

BRAY, T.; MALER, E.; PAOLI, J. **Extensible markup language (xml) 1.0**. Sperberg-McQueena, France, n.2, 2000.

COCA-COLA, H. **The Coca-Cola Virtual Representator.** Publicação Eletrônica, France, n.2, 2002.

CONPET, 2008. **Publicação eletrônica**. Disponível em: <http://www.ed.conpet.gov.br/converse.php>
Acesso em: 18 abr. 2014.

NEVES, A. M. M.; BARROS, F. A. **Iaiml: Um mecanismo para tratamento de intenção**. SBC, Recife, Pe, v.3, n.355, 2005.

CORRÊA, A. **Robô de conversação aplicado à educação à distância como tutor inteligente.** InfoTec, Universidade Federal do Rio Grande do Sul, n.1, p.Acervo Biblioteca RS, 2001.

CASSELL, J. **Genderizing human-computer interaction**. Hillsdale, NJ, USA, n.401, p.Erlbaum Associates, 2003.

LEONHARDT, M. **ELEKTRA: Um *Chatterbot* para Uso em Ambiente Educacional.** Revista Renote, São Paulo, n.2, http://seer.ufrgs.br/renote/article/view/14336, 2005.

FEITOSA, R. M. **PROPOSTA DE ASSISTENTE DE DESKTOP PARA APOIO AO USUÁRIO**. 2008. 56f. Monografia (Bacharelado em Inteligência Artificial) - UNIVERSIDADE FEDERAL DO PARÁ, Belém, PA, 2008.

FILHO, E. C. P. **O Uso do Processamento de Linguagem Natural na Construção de *Chatterbot***. 2009. 58f. Monografia (Bacharelado em Inteligência Artificial) - UNIVERSIDADE FEDERAL DE GOIÁS - UFG, Catalão, GO, 2009.

FINNIN, T.; FRITZSON, R. **A Language and Protocol for Knowledge and Information Exchange**. Technical Report CS-94-02., University of Maryland, UMBC, n.2.Maryland: Computer Science Department, 1994.

FONER, L. N. **Entertaining agents: A sociological case study. In Johnson, W. L.** 2ª. ed. CA, USA: Proceedings of the First International Conference on Au-, 1997. 122p.

FORBES. **Invasion of the Virbots**, September 18, 2000. Publicação eletrônica. Disponível em: http://www.forbes.com/global/2000/0918/0318118a_print.html.
Acesso em 30 mar. 2014.

GALVÃO, A. **Persona-AIML: Uma Arquitetura para Desenvolver *Chatterbot* com Personalidade**. 2003. 74f. Monografia (Bacharelado em Inteligência Artificial) - Centro de Informática, UFPE, Recife, 2003.

GARNER, R.; NATHAN, P. X. **he Evolution of JFRED**. Loebner, Boston, EUA, n.2, 1997.

GIANFORTE, G. **The Insider's Guide to Customer Service on the Web: Ten Secrets for Successful E-Service.** CRM Project Volume 3. Montgomery Research Inc., EUA, n.3, 2002.

KRISTOL, D.; MONTULLI, L. **RFC 2109: HTTP State Management Mechanism**. RFC, Ohio, EUA, n.9, 1997.

MAULDIN, M. L. ***Chatterbot,* tinymuds, and the turing test: Entering the Loebner prize competition.** Lazytoad, Boston, EUA, v.1, n.1, p.Revista Lazytoad, 1999.

NORVIG, P.; RUSSEL, S. **Inteligência Artificial - Tradução da Segunda Edição**. 2ª. ed. São Paulo: Elsevier, 2002. 326p.

PEREIRA, S. L. **Processamento da Linguagem Natural**. 2007. 13f. Monografia (Bacharelado em Inteligência Artificial) - USP, São Paulo, 2007.

RABUSKE, R. A. **Inteligência Artificial**. 1995. 98f. Monografia (Bacharelado em **Inteligência Artificial**) - UFSC, Florianopolis, SC, 1995.

RICH, E. **Inteligência Artificial**. 2ª. ed. Boston, EUA: Makron Books, 1995. 100p.
SGANDERLA, R. B. **"BonoBOT: Um *Chatterbot* para Interação com Usuários em um Sistema Tutor**. 2003. 98f. Dissertação (Mestrado em Inteligência Artificial) - XIV Simpósio Brasileiro de Informática na Educação, Rio de Janeiro, 2003.

SHAW, E.; JOHNSON, W. L. **Using Agents to Overcome Deficiencies in Web-Based Courseware**. Workshop on Pedagogical Agents, Kobe, Japan, n.1, p.Workshop on Pedagogical Agents, 1997.

TEIXEIRA, S. ***Chatterbot*:uma proposta para a construção de bases de conhecimento.** Multicast, São Paulo, SP, n.13, 2005.

TURING, A. M. **Computing machinery and intelligence. MIND: A Quarterly Review of Pyschology and Philosophy**. ISSN 0026-4423, Boston, EUA, v.59, n.236, 1950.

WALLACE, R. **A.L.I.C.E. Artificial Intelligence Foundation**. The Anatomy of A.L.I.C.E, Inc, 2007, n.13, 2007.

WEIZENBAUM, J. E. **A Computer Program for the Study of Natural Language Communication Between Man and Machine**. 1966. 45f. Monografia (Bacharelado em Inteligência Artificial) - C9, Boston, EUA, 1966.

WHALEN, T. **Computational Behaviorism Applied To Natural Language.** Communications Research Centre, EUA, n.3, 1996.

Aiml implementations. A.L.I.C.E Disponível em: <http://www.alicebot.org/downloads/programs.html> Acesso em: 22 jun. 2013

Neuromedia. **How to make a bot,** 1999. Disponível em: <http://www.neurostudios.com/html/documentation/Creation_Tips.htm> Acesso em: 28 set. 2013

OLIVEIRA, Fabio A.D. **Processamento de linguagem natural: princípios básicos e a implementação de um analisador sintático de sentenças da língua portuguesa, 2007.** Disponível em: <http://www.inf.ufrgs.br/procpar/disc/cmp135/trabs/992/Parser/parser.html> Acesso em: 17 out. 2013

Apêndice A – Form Principal

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-02-18T16:58:23
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = ProjetoTCC
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

RESOURCES += \
    Fundo.qrc

OTHER_FILES += \
    my_app.py
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_widget_objectNameChanged(const QString &objectName);

    void on_bt_enviar_clicked();

    void on_graphicsView_2_rubberBandChanged(const QRect &viewportRect,
const QPointF &fromScenePoint, const QPointF &toScenePoint);

    void on_graphicsView_2_objectNameChanged(const QString &objectName);

private:
    Ui::MainWindow *ui;
```

```
};

#endif // MAINWINDOW_H


#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Apêndice B – Arquivo XML Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>762</width>
    <height>490</height>
   </rect>
  </property>
  <property name="whatsThis">
   <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;img
src=&quot;:/imagens/UFPI2.jpg&quot;/&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt
;</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>120</x>
      <y>20</y>
      <width>231</width>
      <height>41</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>8</pointsize>
      <weight>75</weight>
      <bold>true</bold>
     </font>
```

```
 </property>
 <property name="text">
  <string>UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI</string>
 </property>
</widget>
<widget class="QLabel" name="label_2">
 <property name="geometry">
  <rect>
   <x>120</x>
   <y>40</y>
   <width>321</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>8</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>CAMPUS SENADOR HELVIDIO NUNES DE BARROS - CSHNB</string>
 </property>
</widget>
<widget class="QLabel" name="label_3">
 <property name="geometry">
  <rect>
   <x>120</x>
   <y>60</y>
   <width>301</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>8</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>BACHARELADO EM SISTEMAS DE INFORMAÇÃO</string>
 </property>
</widget>
<widget class="QLineEdit" name="msg1">
 <property name="geometry">
  <rect>
   <x>210</x>
   <y>200</y>
   <width>261</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLineEdit" name="msg2">
 <property name="geometry">
  <rect>
   <x>210</x>
   <y>250</y>
   <width>261</width>
```

```
    <height>31</height>
   </rect>
  </property>
 </widget>
 <widget class="QLabel" name="label_4">
  <property name="geometry">
   <rect>
    <x>130</x>
    <y>210</y>
    <width>71</width>
    <height>20</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>9</pointsize>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
  <property name="text">
   <string>Usuário diz:</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_5">
  <property name="geometry">
   <rect>
    <x>110</x>
    <y>260</y>
    <width>101</width>
    <height>20</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>9</pointsize>
    <weight>75</weight>
    <bold>true</bold>
   </font>
  </property>
  <property name="text">
   <string>Systembot diz:</string>
  </property>
 </widget>
 <widget class="QPushButton" name="bt_enviar">
  <property name="geometry">
   <rect>
    <x>490</x>
    <y>230</y>
    <width>75</width>
    <height>31</height>
   </rect>
  </property>
  <property name="text">
   <string>Enviar</string>
  </property>
 </widget>
 <widget class="QGraphicsView" name="graphicsView">
  <property name="geometry">
   <rect>
    <x>10</x>
```

```
        <y>10</y>
        <width>101</width>
        <height>151</height>
       </rect>
      </property>
      <property name="sizePolicy">
       <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
        <horstretch>0</horstretch>
        <verstretch>0</verstretch>
       </sizepolicy>
      </property>
      <property name="mouseTracking">
       <bool>true</bool>
      </property>
      <property name="autoFillBackground">
       <bool>false</bool>
      </property>
      <property name="styleSheet">
       <string notr="true">background-image:
url(:/imagens/UFPI2.jpg);</string>
      </property>
      <property name="interactive">
       <bool>false</bool>
      </property>
     </widget>
     <widget class="QLabel" name="label_6">
      <property name="geometry">
       <rect>
        <x>120</x>
        <y>80</y>
        <width>301</width>
        <height>41</height>
       </rect>
      </property>
      <property name="font">
       <font>
        <pointsize>8</pointsize>
        <weight>75</weight>
        <bold>true</bold>
       </font>
      </property>
      <property name="text">
       <string>TRABALHO DE CONCLUSÃO DE CURSO</string>
      </property>
     </widget>
     <widget class="QLabel" name="label_7">
      <property name="geometry">
       <rect>
        <x>120</x>
        <y>100</y>
        <width>301</width>
        <height>41</height>
       </rect>
      </property>
      <property name="font">
       <font>
        <pointsize>8</pointsize>
        <weight>75</weight>
        <bold>true</bold>
       </font>
      </property>
```

```
 <property name="text">
  <string>DESENVOLVEDOR: ABIMAEL HONÓRIO CORREIA JÚNIOR</string>
 </property>
</widget>
<widget class="QLabel" name="label_8">
 <property name="geometry">
  <rect>
   <x>80</x>
   <y>300</y>
   <width>141</width>
   <height>20</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>9</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>Registro da Conversa:</string>
 </property>
</widget>
<widget class="QLabel" name="resposta">
 <property name="geometry">
  <rect>
   <x>260</x>
   <y>300</y>
   <width>131</width>
   <height>20</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>9</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string/>
 </property>
</widget>
<widget class="QLabel" name="resposta_2">
 <property name="geometry">
  <rect>
   <x>230</x>
   <y>300</y>
   <width>131</width>
   <height>20</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>9</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
```

```
      <string/>
     </property>
    </widget>
    <widget class="QGraphicsView" name="graphicsView_2">
     <property name="geometry">
      <rect>
       <x>600</x>
       <y>170</y>
       <width>101</width>
       <height>151</height>
      </rect>
     </property>
     <property name="styleSheet">
      <string notr="true">background-image:
url(:/imagens/system_robot.jpg);</string>
     </property>
    </widget>
    <zorder>graphicsView</zorder>
    <zorder>label</zorder>
    <zorder>label_2</zorder>
    <zorder>label_3</zorder>
    <zorder>msg1</zorder>
    <zorder>msg2</zorder>
    <zorder>label_4</zorder>
    <zorder>label_5</zorder>
    <zorder>bt_enviar</zorder>
    <zorder>label_6</zorder>
    <zorder>label_7</zorder>
    <zorder>label_8</zorder>
    <zorder>resposta</zorder>
    <zorder>resposta_2</zorder>
    <zorder>graphicsView_2</zorder>
   </widget>
   <widget class="QMenuBar" name="menuBar">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>762</width>
      <height>21</height>
     </rect>
    </property>
   </widget>
   <widget class="QToolBar" name="mainToolBar">
    <attribute name="toolBarArea">
     <enum>TopToolBarArea</enum>
    </attribute>
    <attribute name="toolBarBreak">
     <bool>false</bool>
    </attribute>
   </widget>
   <widget class="QStatusBar" name="statusBar"/>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <resources/>
  <connections/>
</ui>
```

Apêndice C – Código Python para iniciar um diálogo

```
import aiml
import wx
import os, sys


x = aiml.Kernel()
x.learn("saudarbot.aiml")
x.learn("despedidabot.aiml")
x.learn("diversos.aiml")
x.learn("ligacao.aiml")
x.learn("padrao.aiml")
x.learn("personalidade.aiml")
x.learn("risos.aiml")
x.learn("tecnobot.aiml")
x.learn("update.aiml")
x.learn("despedidabot.aiml")

x.setBotPredicate("name", "Systembot")

class WindowClass(wx.Frame):



    def __init__(self, *args, **kwargs):
        super(WindowClass, self).__init__(*args, **kwargs)

        self.basic_gui()

    def basic_gui(self):

        panel = wx.Panel(self) #Instanciando o elemento Panel
        self.CreateStatusBar()

    while True:
        input = raw_input("Usuario")
        response = wx.TextCtrl(panel, pos=(40,10), size=(200, 115)) #Criando Area de Texto
print "Systembot diz", response


        #Botoes
        wx.Button(panel, 1, 'Fechar', (250,100))
        self.SetMenuBar(menuBar)

        self.Bind(wx.EVT_MENU, self.onQuit, exitItem) #Atribuindo evento ao componente
        self.SetSize((400,400)) #Tamanho_Janela
        self.SetTitle('Seja Bem Vindo [' + username + ']') #Nome a ser mostrada na barra de títulos

        self.Bind(wx.EVT_BUTTON, self.onQuit, id=1)
        self.Centre() #Centralizando a tela no centro
        self.Show(True)

    def onQuit(self, e):
        self.Close()
```

```
def main():

    app=wx.App()
    WindowClass(None)
    app.MainLoop()

main()
```

Apêndice D – Classe *AIMLParser*

```
from xml.sax.handler import ContentHandler
from xml.sax.xmlreader import Locator
import sys
import xml.sax
import xml.sax.handler

class AimlParserError(Exception): pass

class AimlHandler(ContentHandler):

        _STATE_OutsideAiml    = 0
        _STATE_InsideAiml     = 1
        _STATE_InsideCategory = 2
        _STATE_InsidePattern  = 3
        _STATE_AfterPattern   = 4
        _STATE_InsideThat     = 5
        _STATE_AfterThat      = 6
        _STATE_InsideTemplate = 7
        _STATE_AfterTemplate  = 8

        def __init__(self, encoding = "UTF-8"):
                self.categories = {}
                self._encoding = encoding
                self._state = self._STATE_OutsideAiml
                self._version = ""
                self._namespace = ""
                self._forwardCompatibleMode = False
                self._currentPattern = ""
                self._currentThat    = ""
                self._currentTopic   = ""
                self._insideTopic = False
                self._currentUnknown = "" # the name of the current unknown element

                self._skipCurrentCategory = False
                self._numParseErrors = 0
                self._validInfo = self._validationInfo101
                self._foundDefaultLiStack = []
                self._whitespaceBehaviorStack = ["default"]
                self._elemStack = []
                self._locator = Locator()
```

```
                self.setDocumentLocator(self._locator)

        def getNumErrors(self):
                "Return the number of errors found while parsing the current document."
                return self._numParseErrors
        def setEncoding(self, encoding):
                self._encoding = encoding

        def _location(self):
                line = self._locator.getLineNumber()
                column = self._locator.getColumnNumber()
                return "(line %d, column %d)" % (line, column)

        def _pushWhitespaceBehavior(self, attr):

                assert len(self._whitespaceBehaviorStack) > 0, "Whitespace behavior stack should
never be empty!"
                try:
                        if attr["xml:space"] == "default" or attr["xml:space"] == "preserve":
                                self._whitespaceBehaviorStack.append(attr["xml:space"])
                        else:
                                raise AimlParserError, "Invalid value for xml:space attribute
"+self._location()
                except KeyError:
                        self._whitespaceBehaviorStack.append(self._whitespaceBehaviorStack[-1])

        def startElementNS(self, name, qname, attr):
                print "QNAME:", qname
                print "NAME:", name
                uri,elem = name
                if (elem == "bot"): print "name:", attr.getValueByQName("name"), "a'ite?"
                self.startElement(elem, attr)
                pass

        def startElement(self, name, attr):

                if self._currentUnknown != "":
                        return

                if self._skipCurrentCategory:
                        return

                try: self._startElement(name, attr)
                except AimlParserError, msg:
                        sys.stderr.write("PARSE ERROR: %s\n" % msg)

                        self._numParseErrors += 1
                                if self._state >= self._STATE_InsideCategory:
                                self._skipCurrentCategory = True

        def _startElement(self, name, attr):
                if name == "aiml":
                        if self._state != self._STATE_OutsideAiml:
                                raise AimlParserError, "Unexpected <aiml> tag "+self._location()
                        self._state = self._STATE_InsideAiml
                        self._insideTopic = False
                        self._currentTopic = u""
                        try: self._version = attr["version"]
                        except KeyError:
```

```
                        self._forwardCompatibleMode = (self._version != "1.0.1")
                        self._pushWhitespaceBehavior(attr)
                elif self._state == self._STATE_OutsideAiml:
                        return
                elif name == "topic":
                        if (self._state != self._STATE_InsideAiml) or self._insideTopic:
                                raise AimlParserError, "Unexpected <topic> tag"+self._location()
                        try: self._currentTopic = unicode(attr['name'])
                        except KeyError:
                                raise AimlParserError, "Required \"name\" attribute missing in <topic>
element "+self._location()
                        self._insideTopic = True
                elif name == "category":
                        if self._state != self._STATE_InsideAiml:
                                raise AimlParserError, "Unexpected <category> tag "+self._location()
                        self._state = self._STATE_InsideCategory
                        self._currentPattern = u""
                        self._currentThat = u""
                        if not self._insideTopic: self._currentTopic = u"*"
                        self._elemStack = []
                        self._pushWhitespaceBehavior(attr)
                elif name == "pattern":
                        if self._state != self._STATE_InsideCategory:
                                raise AimlParserError, "Unexpected <pattern> tag "+self._location()
                        self._state = self._STATE_InsidePattern
                elif name == "that" and self._state == self._STATE_AfterPattern:
                        self._state = self._STATE_InsideThat
                elif name == "template":
                        if self._state not in [self._STATE_AfterPattern, self._STATE_AfterThat]:
                                raise AimlParserError, "Unexpected <template> tag "+self._location()
                        if self._state == self._STATE_AfterPattern:
                                self._currentThat = u"*"
                        self._state = self._STATE_InsideTemplate
                        self._elemStack.append(['template',{}])
                        self._pushWhitespaceBehavior(attr)
                elif self._state == self._STATE_InsidePattern:
                        if name == "bot" and attr.has_key("name") and attr["name"] == u"name":
                                self._currentPattern += u" BOT_NAME "
                        else:
                                raise AimlParserError, ("Unexpected <%s> tag " %
name)+self._location()
                elif self._state == self._STATE_InsideThat:
                        if name == "bot" and attr.has_key("name") and attr["name"] == u"name":
                                self._currentThat += u" BOT_NAME "
                        else:
                                raise AimlParserError, ("Unexpected <%s> tag " %
name)+self._location()
                elif self._state == self._STATE_InsideTemplate and self._validInfo.has_key(name):
                        attrDict = {}
                        for k,v in attr.items():
                                attrDict[k.encode(self._encoding)] = unicode(v)
                        self._validateElemStart(name, attrDict, self._version)
                        self._elemStack.append([name.encode(self._encoding),attrDict])
                        self._pushWhitespaceBehavior(attr)
                        if name == "condition":
                                self._foundDefaultLiStack.append(False)
                else:
                        if self._forwardCompatibleMode:
                                self._currentUnknown = name
                        else:
```

```
                                raise AimlParserError, ("Unexpected <%s> tag " %
name)+self._location()

        def characters(self, ch):
                if self._state == self._STATE_OutsideAiml:
                        return
                if self._currentUnknown != "":
                        return
                if self._skipCurrentCategory:
                        return
                try: self._characters(ch)
                except AimlParserError, msg:
                        sys.stderr.write("PARSE ERROR: %s\n" % msg)
                        self._numParseErrors += 1
                        if self._state >= self._STATE_InsideCategory:
                                self._skipCurrentCategory = True

        def _characters(self, ch):
                text = unicode(ch)
                if self._state == self._STATE_InsidePattern:
                        self._currentPattern += text
                elif self._state == self._STATE_InsideThat:
                        self._currentThat += text
                elif self._state == self._STATE_InsideTemplate:
                        try:
                                parent = self._elemStack[-1][0]
                                parentAttr = self._elemStack[-1][1]
                                required, optional, canBeParent = self._validInfo[parent]
                                nonBlockStyleCondition = (parent == "condition" and not
(parentAttr.has_key("name") and parentAttr.has_key("value")))
                                if not canBeParent:
                                        raise AimlParserError, ("Unexpected text inside <%s>
element "%parent)+self._location()
                                elif parent == "random" or nonBlockStyleCondition:
if len(text.strip()) == 0:
                        return
                        else:
                raise AimlParserError, ("Unexpected text inside <%s> element
"%parent)+self._location()
                        except IndexError:
                                raise AimlParserError, "Element stack is empty while validating text
"+self._location()
                        try: textElemOnStack = (self._elemStack[-1][-1][0] == "text")
                        except IndexError: textElemOnStack = False
                        except KeyError: textElemOnStack = False
                        if textElemOnStack:
                                self._elemStack[-1][-1][2] += text
                        else:
                                self._elemStack[-1].append(["text", {"xml:space":
self._whitespaceBehaviorStack[-1]}, text])
                else:
                        pass

        def endElementNS(self, name, qname):
                uri, elem = name
                self.endElement(elem)

        def endElement(self, name):
                """Wrapper around _endElement which catches errors in _characters()
                and keeps going.
```

```
                """
                if self._state == self._STATE_OutsideAiml:
                        return
                if self._currentUnknown != "":
                        if name == self._currentUnknown:
                                self._currentUnknown = ""
                        return
                if self._skipCurrentCategory:
                        if name == "category":
                                self._skipCurrentCategory = False
                                self._state = self._STATE_InsideAiml
                        return
                try: self._endElement(name)
                except AimlParserError, msg:
                        sys.stderr.write("PARSE ERROR: %s\n" % msg)
                        self._numParseErrors += 1 # increment error count
                        if self._state >= self._STATE_InsideCategory:
                                self._skipCurrentCategory = True

        def _endElement(self, name):
                if name == "aiml":
                        # </aiml> tags are only legal in the InsideAiml state
                        if self._state != self._STATE_InsideAiml:
                                raise AimlParserError, "Unexpected </aiml> tag "+self._location()
                        self._state = self._STATE_OutsideAiml
                        self._whitespaceBehaviorStack.pop()
                elif name == "topic":
                        if self._state != self._STATE_InsideAiml or not self._insideTopic:
                                raise AimlParserError, "Unexpected </topic> tag "+self._location()
                        self._insideTopic = False
                        self._currentTopic = u""
                elif name == "category":
                        if self._state != self._STATE_AfterTemplate:
                                raise AimlParserError, "Unexpected </category> tag "+self._location()
                        self._state = self._STATE_InsideAiml
and
                        key = (self._currentPattern.strip(),
self._currentThat.strip(),self._currentTopic.strip())
                        self.categories[key] = self._elemStack[-1]
                        self._whitespaceBehaviorStack.pop()
                elif name == "pattern":
                        if self._state != self._STATE_InsidePattern:
                                raise AimlParserError, "Unexpected </pattern> tag "+self._location()
                        self._state = self._STATE_AfterPattern
                elif name == "that" and self._state == self._STATE_InsideThat:

                        self._state = self._STATE_AfterThat
                elif name == "template":
                        if self._state != self._STATE_InsideTemplate:
                                raise AimlParserError, "Unexpected </template> tag "+self._location()
                        self._state = self._STATE_AfterTemplate
                        self._whitespaceBehaviorStack.pop()
                elif self._state == self._STATE_InsidePattern:
                        if name not in ["bot"]:
                                raise AimlParserError, ("Unexpected </%s> tag " %
name)+self._location()
                elif self._state == self._STATE_InsideThat:
                        if name not in ["bot"]:
                                raise AimlParserError, ("Unexpected </%s> tag " %
```

```
name)+self._location()
                elif self._state == self._STATE_InsideTemplate:
                        if elem[0] == "condition": self._foundDefaultLiStack.pop()
                else:
                        # Unexpected closing tag
                        raise AimlParserError, ("Unexpected </%s> tag " % name)+self._location()
        _validationInfo101 = {
                "bot":                  ( ["name"], [], False ),
                "condition":    ( [], ["name", "value"], True ), # can only contain <li> elements
                "date":         ( [], [], False ),
                "formal":       ( [], [], True ),
                "gender":       ( [], [], True ),
                "get":          ( ["name"], [], False ),
                "gossip":                       ( [], [], True ),
                "id":           ( [], [], False ),
                "input":        ( [], ["index"], False ),
                "javascript":       ( [], [], True ),
                "learn":        ( [], [], True ),
                "li":           ( [], ["name", "value"], True ),
                "lowercase":    ( [], [], True ),
                "person":       ( [], [], True ),
                "person2":      ( [], [], True ),
                "random":       ( [], [], True ), # can only contain <li> elements
                "sentence":     ( [], [], True ),
                "set":          ( ["name"], [], True),
                "size":         ( [], [], False ),
                "sr":           ( [], [], False ),
                "srai":         ( [], [], True ),
                "star":         ( [], ["index"], False ),
                "system":       ( [], [], True ),
                "template":                     ( [], [], True ), # needs to be in the list because it can be a
parent.
                "that":         ( [], ["index"], False ),
                "thatstar":     ( [], ["index"], False ),
                "think":        ( [], [], True ),
                "topicstar":    ( [], ["index"], False ),
                "uppercase":    ( [], [], True ),
                "version":      ( [], [], False ),
        }

        def _validateElemStart(self, name, attr, version):
                required, optional, canBeParent = self._validInfo[name]
                for a in required:
                        if a not in attr and not self._forwardCompatibleMode:
                                raise AimlParserError, ("Required \"%s\" attribute missing in <%s>
element " % (a,name))+self._location()
                for a in attr:
                        if a in required: continue
                        if a[0:4] == "xml:": continue # attributes in the "xml" namespace can appear
anywhere
                        if a not in optional and not self._forwardCompatibleMode:
                                raise AimlParserError, ("Unexpected \"%s\" attribute in <%s> element
" % (a,name))+self._location()
                if name in ["star", "thatstar", "topicstar"]:
                        for k,v in attr.items():
                                if k == "index":
                                        temp = 0
                                        try: temp = int(v)
                                        except:
                                                raise AimlParserError, ("Bad type for \"%s\" attribute
```

```
(expected integer, found \"%s\") " % (k,v))+self._location()
                                        if temp < 1:
                                                raise AimlParserError, ("\"%s\" attribute must have
non-negative value " % (k))+self._location()
                        try:
                                parent = self._elemStack[-1][0]
                                parentAttr = self._elemStack[-1][1]
                        except IndexError:
                                raise AimlParserError, ("Element stack is empty while validating <%s> " %
name)+self._location()
                        required, optional, canBeParent = self._validInfo[parent]
                        nonBlockStyleCondition = (parent == "condition" and not (parentAttr.has_key("name")
and parentAttr.has_key("value")))
                        if not canBeParent:
                                raise AimlParserError, ("<%s> elements cannot have any contents
"%parent)+self._location()
                        elif (parent == "random" or nonBlockStyleCondition) and name!="li":
                                raise AimlParserError, ("<%s> elements can only contain <li> subelements
"%parent)+self._location()
                        elif name=="li":
                                if not (parent=="random" or nonBlockStyleCondition):
                                        raise AimlParserError, ("Unexpected <li> element contained by <%s>
element "%parent)+self._location()
                                if nonBlockStyleCondition:
                                        if parentAttr.has_key("name"):
                                                if len(attr) == 0:
                                                        if self._foundDefaultLiStack[-1]:
                                                                raise AimlParserError, "Unexpected default
<li> element inside <condition> "+self._location()
                                                        else:
                                                                self._foundDefaultLiStack[-1] = True
                                                elif len(attr) == 1 and attr.has_key("value"):
                                                        pass
                                                else:
                                                        raise AimlParserError, "Invalid <li> inside single-
predicate <condition> "+self._location()
                                        elif len(parentAttr) == 0:
                                                if len(attr) == 0:
                                                        if self._foundDefaultLiStack[-1]:
                                                                raise AimlParserError, "Unexpected default
<li> element inside <condition> "+self._location()
                                                        else:
                                                                self._foundDefaultLiStack[-1] = True
                                                elif len(attr) == 2 and attr.has_key("value") and
attr.has_key("name"):
                                                        pass
                                                else:
                                                        raise AimlParserError, "Invalid <li> inside multi-
predicate <condition> "+self._location()
                        return True

def create_parser():
        """Create and return an AIML parser object."""
        parser = xml.sax.make_parser()
        handler = AimlHandler("UTF-8")
        parser.setContentHandler(handler)
        parser.setFeature(xml.sax.handler.feature_namespaces, True)
        return parser
```