Desenvolvimento de um *chatbot* para auxiliar o ensino de Espanhol como Língua Estrangeira

LEANDRO PADILHA FERREIRA[1]
JOAQUIM QUINTEIRO UCHÔA[2]

[1]E.M.E.F. Maria Angélica V. L. Campello, CLES / SMEC - Rio Grande (RS)
leandro@androle.pro.br
[2]Curso ARL - DCC / UFLA - Cx Postal 3037 - CEP 37200-000 Lavras (MG)
joukim@ginux.ufla.br

Resumo: *Neste artigo propõe-se a criação de um* chatbot *para auxiliar o ensino de língua estrangeira. São citadas algumas das tecnologias disponíveis para fazê-lo e opta-se por usar a linguagem AIML e um módulo escrito na linguagem Python (PyAIML), capaz de interpretar etiquetas AIML, para criar um* chatbot *simples. O artigo explica como criar um pequeno programa em Python para funcionar como* chatbot*. Comenta-se como realizar a integração entre o* chatbot *e o sintetizador de voz Festival e são indicadas referências* on-line *sobre AIML, bem como a página do projeto Esteban,* chatbot *que é resultado da proposta do artigo.*

Palavras-Chave: *Espanhol, Software Livre, ensino, inteligência artificial,* chatbot*, processamento da linguagem natural.*

1 Introdução

Estudantes de Espanhol como Língua Estrangeira (ELE) não têm um professor à sua disposição o tempo todo. Normalmente esses alunos são expostos ao idioma durante os horários das aulas, quando têm contato com o professor e os colegas.

Alunos de ELE exercitariam mais o seu espanhol se pudessem praticar conversação na língua-meta diariamente. Infelizmente nem sempre é possível reunir-se com colegas para praticar, e as aulas de língua estrangeira normalmente ocorrem uma ou duas vezes por semana.

Uma solução para esse problema seria o uso da internet para permitir a comunicação entre estudantes de ELE. Algumas instituições de ensino mantém laboratórios de informática com acesso à internet, o que permite esse contato dos alunos entre si e entre alunos e falantes nativos do idioma em estudos.

Os falantes nativos contatados pelos alunos podem nem sempre estar dispostos a ajudar e o *input* fornecido por eles foge completamente do controle dos professores. Isso torna de fato inconveniente incentivar alunos pré-adolescentes e adolescentes a conversar com qualquer pessoa que conheçam na internet, pois esses desconhecidos podem falar de temas inadequados, ou ainda ter más intenções de algum tipo.

Um *chatbot*, uma entidade de inteligência artificial capaz de simular um falante humano através do Processamento da Linguagem Natural (PLN), poderia oferecer aos alunos a chance de praticar sempre e quando desejarem. Ademais, tal entidade ofereceria as seguintes vantagens:

- Estar disponível mesmo sem acesso à internet;
- Oferecer um escopo controlado de informações, evitando os perigos de expor os alunos a falantes com más intenções;
- Poder ser programado para tratar de variados assuntos e temas, podendo ser usado para que o aluno discuta conteúdos relacionados a outras disciplinas usando a língua-meta;
- Poder motivar os alunos, principalmente os mais tímidos, a relacionar-se com o computador, pois de certa forma humaniza a máquina, simulando afetuosidade;
- Poder ser integrado a um sintetizador de voz, gerando respostas em voz alta, o que é extremamente motivador para os alunos, desmistificando e tornando mais amigável o computador;
- Permitir aos alunos que possuam um computador a possibilidade de ter o programa instalado no seu equipamento, e com isso praticar sempre que desejarem.

Um *chatbot* também apresenta algumas desvantagens, sendo a principal delas o fato de que é um programa de computador e não é capaz de pensar e aprender da mesma forma que um ser humano o faz. Como a quantidade de sentenças que um ser humano pode produzir é praticamente infinita, dificilmente um *software* será capaz de lidar com todos os tipos de *input* fornecido pelos usuários.

Porém, apesar das limitações, esse tipo de programa não pode ser desprezado como ferramenta auxiliar no ensino, não só de ELE, como também de outras disciplinas. O potencial para o uso dessa tecnologia é enorme não só na área da educação, como também para qualquer área que necessite de consulta à bases de dados, como atendimento ao cliente, suporte *on-line*, etc. Conforme comentado em (PALAZZO, 1997): "O PLN é da maior importância para o desenvolvimento de ferramentas para a comunicação homem-máquina em geral e para a construção de interfaces SBCs em particular". Por SBCs Palazzo refere-se a Sistemas Baseados em Conhecimento.

Este artigo pretende apresentar uma possível proposta nesse contexto. Não se pretende aqui criar algo inovador na área de pesquisa em Inteligência Artificial, e sim ilustrar como é possível facilmente criar um *chatbot* e torná-lo um projeto de Software Livre[1] aberto a outros interessados.

2 *Chatbots* e Inteligência Artificial

Inteligência artificial, ou IA, é um ramo do conhecimento humano que busca reproduzir a mente humana usando elementos computacionais. Não faz parte do escopo do presente artigo definir as várias correntes que formam o estudo de IA nos dias atuais, ou o seu desenvolvimento através da história. Para maiores informações sobre o tema, pode-se consultar (RUSSELL; NORVIG, 2002).

[1] `http://www.fsf.org`

O *chatbot* proposto é uma entidade de IA, destinado a processar linguagem natural. Em (ROBIN, 2001) tem-se uma definição de PLN:

> O Processamento de Linguagem Natural (PLN) é um ramo da Inteligência Artificial (IA) que tem por objetivo interpretar e gerar textos em uma língua natural (*e.g.*, Português, Inglês, Francês, Espanhol, etc).
>
> O PLN é genuinamente multi-disciplinar, congregando, principalmente, estudos nas áreas de Ciência da Computação, Lingüística e Ciências Cognitivas.

A partir disso, pode-se definir que o *chatbot* deve ser capaz de interpretar e gerar textos na língua espanhola e ter uma personalidade definida. Criar uma personalidade para o *chatbot* pode gerar um resultado mais produtivo, sobretudo por ter como público alvo jovens estudantes, pois a simulação de um ser humano fica mais consistente e mais lúdica, além de facilitar a coerência na leitura das respostas do programa. Definir o que é a personalidade é algo complexo. Pode-se ver em (HALL; LINDZEY, 1984):

> Para nós *a personalidade é definida por conceitos empíricos particulares que são uma parte da teoria da personalidade empregada pelo observador.*[2] A personalidade realmente compõe-se de um conjunto de valores ou termos descritivos usados para caracterizar o indivíduo estudado de acordo com as variáveis ou dimensões que ocupam posição central na teoria adotada.
>
> (...)Noutras palavras, é impossível definir personalidade sem a aceitação de uma linha teórica de referência dentro da qual a personalidade será pesquisada.

Estudos posteriores serão necessários para dar uma personalidade coerente ao *chatbot*. Por enquanto, apenas o seu gênero, nome e a sua língua estão definidos. É um recurso importante definir todos os elementos que formariam uma personalidade humana nas unidades de respostas do programa, tais como preferências, senso de humor, reação à ofensas ou elogios, etc. Todos esses elementos reunidos podem reforçar nos usuários a sensação de estar efetivamente comunicando-se com um ser humano, mesmo sabendo tratar-se de um programa de computador desde o início da interação. Como explicado em (PASSOS; PASSOS, 1990) não é apenas a língua a responsável pela comunicação:

> A função essencial da linguagem é a comunicação. Essa comunicação, entretanto, realiza-se pela interação da língua com outros processos mentais, tais como o conhecimento de fatos, o conhecimento do mundo, a experiência individual, i.e., a comunicação é resultante da inter-relação da língua com todos os fenômenos cognitivos da mente humana.

Assim, depois de algum tempo de interação o usuário ficará saturado com determinadas características das respostas do *chatbot* e formará esquemas mentais que ajudarão a compor a coerência das respostas do programa. Conforme comentado em (KOCH; TRAVAGLIA, 1996), pode-se definir a coerência textual como:

[2]Grifo no original

> ... a coerência está diretamente ligada à possibilidade de se estabelecer um sentido para o texto, ou seja, ela é o que faz com que o texto faça sentido para os usuários, devendo, portanto, ser entendida como um princípio de interpretabilidade, ligada à inteligibilidade do texto numa situação de comunicação e à capacidade que o receptor tem para calcular o sentido deste texto. Este sentido, evidentemente, deve ser do todo, pois a coerência é global.

3 Proposta de desenvolvimento

Chatbot ou *chatterbot* é um programa de computador criado com o propósito de simular a habilidade de conversação de um ser humano. Apesar de ser um propósito simples de definir, a implementação de um programa de computador capaz de atingir esse propósito é algo extremamente complexo. Como até hoje não foi possível duplicar a mente humana, capaz de raciocínio e criatividade, os programas de computador que tentam simulá-la geralmente usam recursos da psicologia para "fingir" que são seres humanos.

Expedientes como devolver o *input* do usuário como uma pergunta, reagir de acordo com sentenças armazenadas em um banco de dados, analisar gramaticalmente as frases, registrar as respostas prévias e acessar aleatoriamente essas respostas, terminam por simular um ser humano real com razoável sucesso. Há pelo menos duas maneiras de enfocar o problema da criação de *chatbots*, considerando-se apenas os conhecimentos de informática necessários:

- Usar Prolog, ou outra linguagem de programação em lógica, e criar um *chatbot* a partir do zero, ou ainda adaptar uma versão existente de algum programa desse tipo. Requer conhecimentos elaborados de programação de computadores e de programação em lógica;
- Usar uma linguagem de marcação desenvolvida especialmente para esse propósito como a *Artificial Intelligence Markup Language* (AIML). A especificação da AIML encontra-se em (BUSH, 2001). AIML é virtualmente independente de linguagem de programação. Também é necessário um processador de AIML, escrito na linguagem que mais convém ao usuário (mais informações sobre AIML podem ser encontradas na seção 3.1). Essa opção permite um rápido desenvolvimento sem a necessidade de aprender recursos sofisticados de programação. Interessados podem aprender facilmente a escrever arquivos AIML, mesmo que não tenham conhecimentos técnicos em informática.

Pode-se também criar *chatbots* usando linguagens procedimentais, como Perl ou C. Neste trabalho optou-se pelo AIML pela série de facilidades que ele oferece ao desenvolvedor (ver seção 3.1). Busca-se aqui desenvolver um *chatbot* que funcione, que seja simples e que possa ser modificado por pessoas relativamente leigas em informática. Ou seja, que até pessoas que não sejam hábeis em alguma linguagem de programação possam modificá-lo.

Esse é o aspecto principal que fez com que o autor escolhesse o AIML, a enorme facilidade de desenvolvimento (um *bot* simples pode ser criado em questão de horas) e de aprendizagem.

A linguagem Python[3] foi escolhida para criar o programa por ser uma linguagem fácil de aprender, bem documentada e com uma enorme quantidade de módulos prontos para uso para ampliar a sua funcionalidade original.

3.1 Projeto ALICE e AIML

O projeto ALICE[4], "Artificial Linguistic Internet Computer Entity", foi desenvolvido pelo doutor Richard S. Wallace. A primeira edição de ALICE foi implementada em 1995 e o programa ganhou o "Loebner Prize"[5] nos anos de 2000, 2001 e 2004.

AIML[6] é baseada em XML. A definição formal de XML está em (YERGEAU, 2004) e é definida em (RAMALHO, 2002) como:

> Uma abreviação de eXtensible Markup Language, ... Assim como a linguagem HTML, ela tem como finalidade marcar um determinado texto que sofrerá algum tipo de processamento.

AIML é possível de ser aprendida inclusive por quem não é programador profissional. Segundo (BUSH, 2001)[7]:

> A Linguagem de Marcação de Inteligência Artificial é uma derivação da XML (Linguagem de Marcação Extensível) que é completamente descrita neste documento. O objetivo é possibilitar que conteúdo baseado em padrões do tipo estímulo-resposta possa ser oferecido, recebido e processado na Web e off-line do mesmo modo que é atualmente possível com HTML e XML. AIML foi desenhada tendo em vista a facilidade de implementação, facilidade de uso por novos usuários, e para inter-operabilidade com XML e XML-derivados, como XHTML.

As etiquetas básicas de AIML são apresentadas na Figura 1. Uma lista mais completa das etiquetas pode ser vista em (BUSH, 2001), e uma lista organizada como uma tabela pode ser vista no *site* de documentação do projeto ALICE[8]. Em (BANACH, 2004) comenta-se como usar algumas das etiquetas AIML para criar um *chatbot*. Em (WALLACE, 2002) pode-se encontrar um *tutorial* de AIML escrito pelo autor da linguagem. Em (RINGATE, 2002), há outro *tutorial* de AIML. Há um manual sobre como adicionar conhecimento ao *chatbot* criado com AIML em (AIMLESS, 2002), e em (WALLACE, 2000), tem-se um artigo sobre como fazer reduções simbólicas no código AIML.

Na Figura 1 vê-se que o arquivo começa com a definição da versão e da codificação do XML usado. A etiqueta principal é <aiml>, é ela que abre e fecha cada arquivo

[3] `http://www.python.org`

[4] `http://alicebot.org`

[5] `http://www.loebner.net/Prizef/loebner-prize.html` O Loebner Prize é uma implementação do Teste de Turing, nome dado ao teste idealizado pelo matemático Alan Turing para testar a capacidade de um programa de computador de convencer um juiz humano de que é outro ser humano.

[6] *Artificial Intelligence Markup Language* ou Linguagem de Marcação de Inteligência Artificial

[7] Tradução nossa

[8] Documentação do projeto `http://www.alicebot.org/documentation` e a tabela em `http://www.alicebot.org/documentation/aiml101.html`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aiml version="1.0.1">
<category>
    <pattern>OI</pattern>
    <template>Oi, tudo bem?</template>
</category>
</aiml>
```

Figura 1: Etiquetas mínimas em AIML

que contém código AIML. Com o atributo *version="1.0.1"* define-se a versão de AIML que está sendo usada. A etiqueta <category> contém uma unidade de informação. Para cada possível pergunta feita ao *chatbot* é necessário criar uma nova categoria. A pergunta que é feita ao *chatbot* fica entre as etiquetas <pattern> e a resposta entre as etiquetas <template>. A informação colocada em <pattern> deve estar em letras maiúsculas. Na hora de usar o *chatbot* não importa se a informação estará em maiúsculas ou minúsculas. Tudo que estiver entre as etiquetas <template> será devolvido pelo programa como resposta à pergunta. Outras etiquetas podem ser usadas para melhorar o desempenho do *chatbot*, como se vê na Figura 2.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aiml version="1.0.1">
<category>
    <pattern>OI</pattern>
    <template>
      <random>
          <li>Oi, tudo bem?</li>
          <li>Como vai você?</li>
          <li>Olá</li>
          <li>Oi</li>
          </random>
    </template>
</category>
<category>
    <pattern>OLÁ</pattern>
    <template>
        <srai>OI</srai>
    </template>
</category>
</aiml>
```

Figura 2: Outras etiquetas AIML

A etiqueta <random> permite que o *chatbot* escolha de forma aleatória o conteúdo de uma das etiquetas <li> como resposta ao <pattern> da categoria. A etiqueta <srai>

chama tudo que estiver nas etiquetas <template> da categoria cujo <pattern> seja igual ao texto que estiver delimitando. No exemplo mostrado ao digitar "Olá" o usuário terá a mesma resposta que teria digitando "Oi" para o *chatbot*. Com isso, evita-se ter de escrever várias categorias semelhantes, ou com respostas iguais. Outra etiqueta importante é a etiqueta <system>. Um exemplo pode ser visto na Figura 3. A etiqueta <system> acessa os programas instalados no sistema operacional em uso; nesse caso o programa *date*, que retorna a hora e a data do sistema em sistemas GNU/Linux. Qualquer comando que poderia ser dado através do *prompt* de comando do GNU/Linux pode ser usado na etiqueta <system>.

```
<category>
    <pattern>QUE HORA ES</pattern>
    <template>
      <system>echo "Son las `date +%H` horas"</system>
    </template>
</category>
```

Figura 3: Uso da etiqueta <system>

Existem vários programas para interpretar AIML. Na página `http://alicebot.org/download`, pode-se encontrar diversos programas capazes disso. Um interpretador é um programa capaz de ler e interpretar as etiquetas do AIML. Qualquer *chatbot* baseado em AIML necessita de um interpretador. Hoje existem interpretadores escritos em Java, PHP, C, Python e outras linguagens.

4 Esteban

Esteban é o *chatbot* criado pelo primeiro autor deste artigo para ilustrar a facilidade de uso do AIML e também para futuramente ser usado como ferramenta de auxílio ao ensino de ELE. Inicialmente, ele conta com uma capacidade rudimentar de responder às perguntas. Espera-se que com o tempo e a ajuda de outros professores a sua *inteligência* possa crescer e possa ser definida uma verdadeira personalidade para o programa.

4.1 Primeiros passos

O primeiro passo para criar um novo *chatbot* baseado em AIML é escolher um interpretador de AIML. Nesse caso o interpretador escolhido foi o PyAIML[9]. Este *software* foi escrito na linguagem Python, e pode ser usado como um módulo da linguagem em programas escritos em Python. Isso quer dizer que o usuário pode acessar o conjunto de classes, métodos e variáveis que fazem parte do módulo e usá-los em seus próprios programas Python.

Depois de obter o PyAIML e instalá-lo no computador que será usado para desenvolvimento, faz-se necessário criar um programa capaz de carregar o módulo capaz de

[9] `http://pyaiml.sourceforge.net`

interpretar AIML (chamado aiml) e as categorias de dados criadas usando-se a linguagem AIML, que são a verdadeira *inteligência* de Esteban.

Optou-se por criar os arquivos com código AIML em um diretório chamado "aiml", e usar a extensão *.aiml para facilitar a identificação e carga desse tipo de arquivo. Cada arquivo *.aiml no diretório aiml/ contém um conjunto de categorias relacionadas entre si.

A Figura 4 mostra fragmentos do código *python* de Esteban. O programa completo é capaz de fazer tratamento de erros básico e gravar registros das conversas entre o usuário e o *chatbot*.

```
#!/usr/bin/python
# encoding: ISO-8859-1
import os
import aiml
k = aiml.Kernel()
k.setTextEncoding("Latin1")
k.learn("aiml/*.aiml")
k.setBotPredicate("name","Esteban")
user_input = "begin"
while user_input != "quit":
    user_input = raw_input("> ")
    answer = k.respond(user_input)
    print answer
    #os.system('echo %s | festival -b --tts --language spanish' %
        answer)
```

Figura 4: Fragmento de código de esteban.py

A seguir as partes principais do código do programa são comentadas:

```
k = aiml.Kernel()
```

Essa linha cria uma instância da classe *Kernel* do PyAIML. Essa é a classe básica para usar o PyAIML.

```
k.setTextEncoding("Latin1")
```

Acessando o método setTextEncoding() da classe Kernel() usando o argumento "Latin1" o programa poderá funcionar em sistemas com codificação diferente de UTF-8, que é a codificação padrão usada pelo PyAIML. Sem usar esse recurso em sistemas que usam codificação ISO-8859-1, os caracteres acentuados não podem ser usados. Não foram feitos testes em máquinas que usem a codificação UTF-8, mas no caso de haver problemas basta transformar essa linha em um comentário com o símbolo #.

```
k.learn("aiml/*.aiml")
```

Esse comando acessa o método *learn* da classe *Kernel* e diz onde estão os arquivos contendo código AIML que devem ser carregados. Dessa forma todos os arquivos que tenham a extensão *.aiml dentro do diretório "aiml" serão carregados.

```
k.setBotPredicate("name","Esteban")
```

O método *setBotPredicate* da classe *Kernel* é usado para registrar o nome do *chatbot*.

```
while user_input != "quit":
    user_input = raw_input("> ")
    answer = k.respond(user_input)
    print answer
        #os.system('echo %s | festival -b --tts --language
            spanish' % answer)
```

Esse laço de repetição não tem fim. Ele repete-se até que o programa seja interrompido com a introdução da palavra 'quit'. É definida a variável *user_input* com o resultado de uma consulta ao usuário. A variável *answer* guardará a resposta do *chatbot* e a seguir o seu conteúdo é impresso na tela. No código completo do programa vê-se ver como é feita a gravação das perguntas/respostas e como ativar o suporte ao sintetizador de voz Festival (ver Seção 4.2).

Uma vez que esse arquivo é gravado com o nome **esteban.py** basta torná-lo executável pelo sistema com o comando **chmod a+x esteban.py** e então chamá-lo à execução como pode ser visto na Figura 5. As frase escritas pelo usuário começam com o símbolo >.

```
$ ./esteban.py
> Hola
Buenos días!
> blabla
WARNING: No match found for input: blabla
> Como te llamas?
No me gustaría decirlo.
> donde vives?
Yo vivo en la memoria de un ordenador.
> soy Leandro
Encantado Leandro!
> donde vives
Yo vivo en la memoria de un ordenador.
Tienes problemas de memoria, verdad?
> quit
Chau, nos vemos pronto
```

Figura 5: Executando Esteban

Na Figura 6 pode-se ver um exemplo do arquivo que registra as conversas entre o interlocutor humano e Esteban. É muito importante manter esse tipo de arquivo de registro, pois é através dele que se poderá ver quais foram as perguntas que Esteban não conseguiu responder e, com essa informação, pode-se criar novas categorias que abarquem também essas perguntas não respondidas.

Aquilo que Esteban não souber responder aparecerá no arquivo de registro como uma linha em branco. Então, basta procurar linhas como *Esteban:* e olhar as perguntas feitas na linha acima (nesse exemplo "blabla") para saber com quais sentenças o programa não

```
================================================
Nueva charla, Fecha: 19/3/2005
Inicio de la charla: 11:51:49
usuario: Hola
Esteban: Buenos días!
usuario: blabla
Esteban:
usuario: Como te llamas?
Esteban: No me gustaría decirlo.
usuario: donde vives?
Esteban: Yo vivo en la memoria de un ordenador.
usuario: soy Leandro
Esteban: Encantado Leandro!
usuario: donde vives
Esteban: Yo vivo en la memoria de un ordenador. Tienes problemas
    de
memoria, verdad?
usuario: quit
Esteban: Chau, nos vemos pronto
Fin de la charla: 11:52:34
================================================
```

Figura 6: Exemplo de registro de conversas

soube lidar. Também é possível personalizar a mensagem escrita por Esteban no caso de não saber responder algo.

```
<category>
    <pattern> * </pattern>
    <template>No comprendo lo que quieres decir.</template>
</category>
```

Nesse exemplo de código AIML, vê-se como criar uma categoria para isso. Desse modo, a cada sentença não compreendida o programa escreveria a frase escolhida e a ocorrência seria armazenado no arquivo de registros.

Com isso o *chatbot* é capaz de funcionar. O código Python usado não é o aspecto principal de Esteban. A verdadeira "inteligência" de Esteban, como já foi dito, reside nos arquivos *.aiml.

Como está em um estágio inicial muito ainda deve ser feito para que ele possa responder a um amplo leque de perguntas. O programa ALICE conta com mais de 40.000 categorias. Esteban atualmente tem menos de 100. O autor espera conseguir ajuda de outros professores de ELE para aumentar a capacidade de resposta do *chatbot*.

4.2 Usando o Festival

O Festival[10] (Festival Speech Synthesis System) é um projeto do "Centre for Speech Technology Research"[11] da Universidade de Edinburgo[12], da Escócia. Para usá-lo como sintetizador de voz basta tê-lo instalado no sistema[13] e descomentar a linha que está em **esteban.py**:

```
#os.system('echo %s | festival -b --tts --language spanish' %
    answer)
```

Para descomentar basta apagar o símbolo #. Esse comando acessa através da linguagem Python os programas do sistema GNU/Linux que está executando Esteban. Nesse caso o comando *echo* envia ao **festival** o que foi respondido pelo *chatbot* (que está armazenado na variável *answer*). E o programa **festival** é executado de modo a reproduzir os dados que recebeu do *echo*, na forma de voz sintetizada.

5 Conclusão

Conclui-se que é possível desenvolver rápida e facilmente um *chatbot* simples usando AIML. E mesmo pessoas não habituadas a programar computadores podem aprender, com algum treinamento mínimo, a criar padrões de resposta para esse *chatbot*, permitindo que professores de todas as áreas possam criar ou ajudar em projetos desse tipo, seja trabalhando sozinhos ou em equipes.

Pode-se também criar modos de acessar programas instalados em um determinado sistema através de comandos em linguagem natural, humanizando mais a relação do homem com a máquina e facilitando o acesso à informática para pessoas idosas ou crianças pequenas. As possibilidades de uso dessa tecnologia são inúmeras.

A licença escolhida para a distribuição de Esteban foi a Creative Commons - GPL (CC-GNU GPL)[14], que é a Licença Geral Pública (GPL), da Free Software Foundation, traduzida ao português brasileiro e com validade legal no Brasil.

Atualmente a página oficial do projeto está no endereço `http://androle.pro.br/esteban`. É uma página simples, que está em contínuo desenvolvimento e serve como meio de divulgação do projeto e da lista de discussão por correio eletrônico criada para os usuários do programa. Pode-se obter uma cópia totalmente funcional de Esteban nesse endereço. De acordo com a necessidade e o crescimento do projeto novas ferramentas poderão ser utilizadas, como, por exemplo, um fórum *on-line*. Também é necessário criar uma documentação mais detalhada para o programa, e sobretudo aumentar a quantidade de categorias existentes nos arquivos AIML. Esteban ainda é um bebê, pouco sabe do mundo.

[10] `http://www.cstr.ed.ac.uk/projects/festival/`

[11] `http://www.cstr.ed.ac.uk/`

[12] `http://www.ed.ac.uk/`

[13] bem como o módulo *Castilian Spanish male speaker*, cujo pacote tem o nome de festvox-ellpc11k em sistemas Debian GNU/Linux

[14] `http://creativecommons.org/licenses/GPL/2.0/`

Referências

AIMLESS, D. *A Tutorial for adding knowledge to your robot*. rev 5. [S.l.], Setembro 2002. Acessado em 26/03/2005. Disponível em: <http://www.pandorabots.com-/pandora/pics/aimless/tutorial.htm>.

BANACH, Z. Construimos nuestro propio bot. *Software 2.0 Extra!*, n. 1, 2004. Acessado em 26/03/2005. Disponível em: <http://www.software20.org/es/attachments/SI_aiml_ES-.pdf>.

BUSH, N. *Artificial Intelligence Markup Language (AIML) Version 1.0.1 A.L.I.C.E. AI Foundation Working Draft*. rev 006. [S.l.], Outubro 2001. Acessado em 26/03/2005. Disponível em: <http://www.alicebot.org/TR/2001/WD-aiml% -/>.

HALL, C. S.; LINDZEY, G. *Teorias da personalidade*. 1. ed. São Paulo: E.P.U, 1984.

KOCH, I. V.; TRAVAGLIA, L. C. *A coerência textual*. 7. ed. São Paulo: Contexto, 1996.

PALAZZO, L. A. M. *Introdução à Programação PROLOG*. 1. ed. Pelotas: EDUCAT, 1997.

PASSOS, C.; PASSOS, M. E. *Princípios de uma gramática modular*. 1. ed. São Paulo: Contexto, 1990.

RAMALHO, J. A. *XML teoria e prática*. São Paulo: Berkeley Brasil, 2002.

RINGATE, T. *AIML Primer*. [S.l.], Janeiro 2002. Contributing Authors: Dr. Richard S. Wallace; Anthony Taylor; Jon Baer; Dennis Daniels. Acessado em 26/03/2005. Disponível em: <http://www.alicebot.org/documentation/aiml-primer.html>.

ROBIN, F. de Almeida Barros e J. Processamento de linguagem natural. *REIC - Revista Eletrônica de Iniciação Científica*, n. II, Novembro 2001. ISN 1519-8219. Acessado em 26/03/2005. Disponível em: <http://www.sbc.org.br/>.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2. ed. New Jersey: Prentice Hall, 2002.

WALLACE, R. S. *Symbolic Reductions in AIML*. [S.l.], Março 2000. Acessado em 26/03/2005. Disponível em: <http://www.alicebot.org/documentation/srai.html>.

WALLACE, R. S. Aiml overview. Acessado em 26/03/2005. 2002. Disponível em: <http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html>.

YERGEAU, F. (Ed.). *Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation*. [S.l.], Fevereiro 2004. Acessado em 26/03/2005. Disponível em: <http://www.w3.org/TR/2004/REC-xml-20040204/>.